HITACHI
Inspire the Next

Cosminexus V11 アプリケーションサーバ アプリケーション設定操作ガイド

操作書

3021-3-J13-50

前書き

■ 対象製品

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の前書きの対象製品の説明を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ 発行

2024 年 2 月　3021-3-J13-50

■ 著作権

変更内容

変更内容(3021-3-J13-50) uCosminexus Application Server 11-40，uCosminexus Client 11-40，uCosminexus Developer 11-40，uCosminexus Service Architect 11-40，uCosminexus Service Platform 11-40

追加・変更内容	変更個所
マニュアル訂正の内容を反映した。	-

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルをお読みになる際の前提情報については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」のはじめにの説明を参照してください。

目次

1 アプリケーション設定操作の概要

この章では，アプリケーション設定操作の目的と位置づけについて説明します。また，アプリケーション設定操作でできることについても説明します。

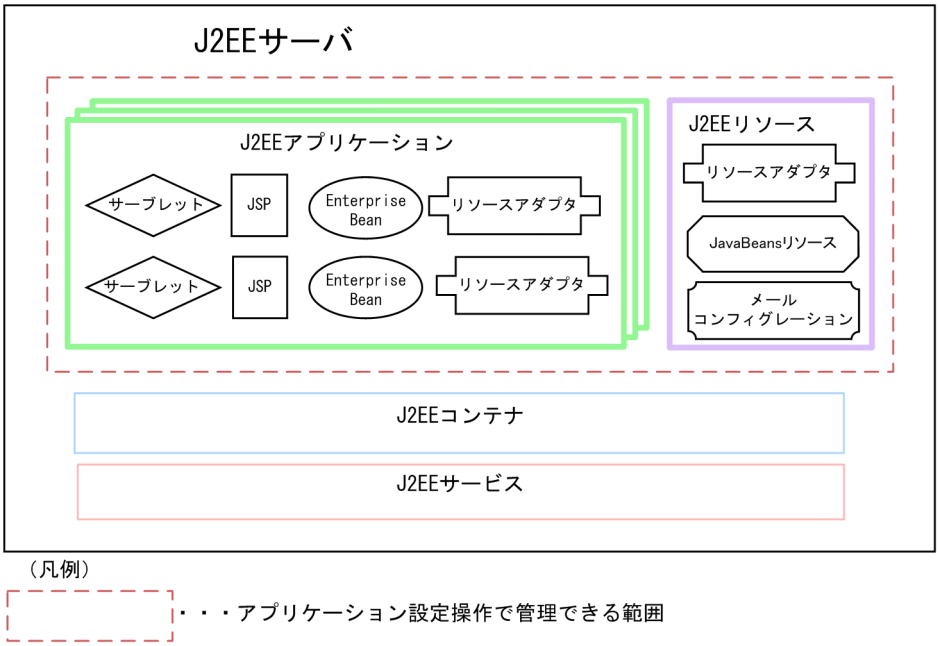
1.1　アプリケーション設定操作の目的

アプリケーション設定操作とは，アプリケーションやリソースを J2EE サーバ上で動作させるために必要な設定およびカスタマイズ操作のことです。次のような場合に実行します。

- アプリケーションサーバシステムの動作環境の構築，運用で，リソースや J2EE アプリケーションを設定する場合
- J2EE アプリケーションの開発で，リソースや J2EE アプリケーションを設定する場合

J2EE サーバの構成およびアプリケーション設定操作で管理するアーキテクチャの範囲を次の図に示します。

図 1-1　アプリケーション設定操作で管理するアーキテクチャの範囲



アプリケーション設定操作はサーバ管理コマンドを使用して実行します。サーバ管理コマンドについては，「2.1　サーバ管理コマンドの機能」を参照してください。

1.2　J2EE リソースの管理

この節では，アプリケーション設定操作での J2EE リソースの管理の概要について説明します。

1.2.1　管理する J2EE リソースの種類

アプリケーション設定操作で管理する J2EE リソースを次に示します。

(1)　リソースアダプタ

リソースアダプタは，次のどちらかの方法で管理します。

- **J2EE リソースアダプタとしてデプロイして使用する。**
  J2EE サーバにインポートしたリソースアダプタを，共有スタンドアロンモジュールとしてデプロイします。その J2EE サーバ上で動作するすべての J2EE アプリケーションで使用できるようになります。J2EE サーバ上にデプロイされたリソースアダプタを J2EE リソースアダプタといいます。
  J2EE リソースアダプタのアプリケーション設定操作については，「1.2.2　リソースアダプタのアプリケーション設定操作」を参照してください。
- **J2EE アプリケーションに含めて使用する。**
  J2EE アプリケーションに含まれるリソースアダプタを，その J2EE アプリケーションで使用できるようにします。
  J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作については，「1.2.3 J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作」を参照してください。

> **注意事項**
>
> J2EE アプリケーションは，これらのどちらのリソースアダプタも使用できます。ただし，同じ表示名のリソースアダプタを同時に使用することはできません。同じ表示名に設定するとエラーとなります。

J2EE リソースアダプタとして使用するリソースアダプタと J2EE アプリケーションに含めて使用するリソースアダプタの比較を次に示します。

表 1-1　管理するリソースアダプタの種類

リソースアダプタ			J2EE リソースアダプタとして使用	J2EE アプリケーションに含めて使用
DB Connector	トランザクションサポートレベル	NoTransaction または LocalTransaction	○	○
		XATransaction	○	×

リソースアダプタ			J2EE リソースアダプタとして使用	J2EE アプリケーションに含めて使用
	クラスタ構成		○	×
TP1 Connector	トランザクションサポートレベル	NoTransaction または LocalTransaction	○	○
		XATransaction	○	×
TP1/Message Queue - Access			○	×
DB Connector for Reliable Messaging および Reliable Messaging			○	×
そのほかのリソースアダプタ			○	○※

（凡例）　○：使用できる　×：使用できない

注※　次のリソースアダプタは，J2EE アプリケーションに含めて管理することはできません。

- トランザクションサポートレベルにグローバルトランザクション管理（XATransaction）が設定されたリソースアダプタ
- ネイティブライブラリを含むリソースアダプタ
- 起動順序の制御が必要なリソースアダプタ

(2)　リソースアダプタ以外の J2EE リソース

リソースアダプタ以外に，次の J2EE リソースを管理します。

- JavaBeans リソース
- メールコンフィグレーション

リソースアダプタ以外の J2EE リソースのアプリケーション設定操作については，「1.2.4　リソースアダプタ以外の J2EE リソースのアプリケーション設定操作」を参照してください。

1.2.2　リソースアダプタのアプリケーション設定操作

J2EE リソースアダプタとしてデプロイして使用するリソースアダプタの，アプリケーション設定操作について説明します。

(1)　リソースアダプタの種類

J2EE リソースアダプタとしてデプロイして使用するリソースアダプタの種類は次のとおりです。

- DB Connector
  JDBC インタフェースを使用してデータベースに接続する場合に使用します。
- DB Connector for Reliable Messaging および Reliable Messaging
  JMS インタフェースを使用してデータベースに接続する場合に使用します。

DB Connector for Reliable Messaging については，マニュアル「Reliable Messaging」の「2.7 DB Connector for Reliable Messaging の機能」を参照してください。Reliable Messaging の管理については，マニュアル「Reliable Messaging」の「2.3　メッセージの管理」を参照してください。

- TP1 Connector
  OpenTP1 の SPP に接続する場合に使用します。
- TP1/Message Queue - Access
  TP1/Message Queue に接続する場合に使用します。
- そのほかのリソースアダプタ
  任意のリソースに接続するためのリソースアダプタとして，Connector 1.5 に準拠したリソースアダプタを使用できます。

(2)　リソースアダプタの管理の流れ

リソースアダプタを管理する基本的な流れを次に示します。

1. リソースアダプタのインポート
2. リソースアダプタのプロパティ定義
3. リソースアダプタのデプロイ
4. J2EE リソースアダプタの接続テスト
5. J2EE リソースアダプタの開始・停止
6. J2EE リソースアダプタのエクスポート

各手順について説明します。

- リソースアダプタのインポート
  リソースアダプタをインポートします。
- リソースアダプタのプロパティ定義
  データベースと接続するための情報を定義じます。
- リソースアダプタのデプロイ
  インポートしたリソースアダプタを J2EE リソースアダプタとしてデプロイします。J2EE リソースアダプタとしてデプロイされたリソースアダプタは，J2EE サーバ上で動作するすべての J2EE アプリケーションから使用できます。
- J2EE リソースアダプタの接続テスト
  接続テストを実施して，J2EE リソースアダプタが正しく動くことを確認します。
- J2EE リソースアダプタの開始・停止
  設定が完了した J2EE リソースアダプタを開始します。また，運用に応じて J2EE リソースアダプタを停止します。

- J2EE リソースアダプタのエクスポート

  J2EE リソースアダプタを，運用に応じてエクスポートします。

  データベースと接続する場合，データベースの設定が必要になります。

  また，リソースアダプタを使用する場合，J2EE アプリケーションでリソースアダプタのリファレンスを解決する必要があります。リソースアダプタのリファレンス解決は，J2EE アプリケーションのカスタマイズで実行します。J2EE アプリケーションのカスタマイズの概要については，「1.3　J2EE アプリケーションの管理」を参照してください。

(3)　リソースアダプタのアプリケーション設定操作

リソースアダプタの管理で使用する，アプリケーション設定操作の機能を次に示します。

表 1-2　リソースアダプタの管理に使用するアプリケーション設定操作

J2EE リソース管理の操作	機能	コマンド
リソースアダプタのインポート	リソースアダプタを J2EE サーバにインポートします。	cjimportres（-type rar 指定）
リソースアダプタのプロパティ定義	デプロイ前のリソースアダプタの属性を取得して，属性ファイルを生成します。	cjgetresprop（-type rar 指定）
	デプロイ前のリソースアダプタの属性を，属性ファイルに指定された値に変更します。	cjsetresprop（-type rar 指定）
	デプロイされている J2EE リソースアダプタの属性を取得して，属性ファイルを生成します。	cjgetrarprop
	デプロイされている J2EE リソースアダプタの属性を，属性ファイルに指定された値に変更します。	cjsetrarprop
リソースアダプタのデプロイ	リソースアダプタをデプロイします。	cjdeployrar
J2EE リソースアダプタの接続テスト	J2EE リソースアダプタの接続テストをします。	cjtestres（-type rar 指定）
J2EE リソースアダプタの開始と停止	J2EE リソースアダプタを開始します。	cjstartrar
	開始されている J2EE リソースアダプタを停止します。	cjstoprar
J2EE リソースアダプタのエクスポート	J2EE サーバ上のリソースアダプタをエクスポートします。	cjexportrar
リソースの一覧表示	インポート済みのリソースアダプタの一覧を表示します。	cjlistres（-type rar 指定）
	J2EE リソースアダプタの一覧を表示します。	cjlistrar
リソースの削除	リソースアダプタを削除します。	cjdeleteres（-type rar 指定）
	J2EE リソースアダプタをアンデプロイします。	cjundeployrar
そのほかの操作	リソースアダプタのコネクションプールの状態を表示します。	cjlistpool

J2EE リソース管理の操作	機能	コマンド
	リソースアダプタのコネクションプールのコネクションを削除します。	cjclearpool
	リソースアダプタの属性をコピーします。	cjcopyres（-type rar 指定）

1.2.3 J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作

J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作について説明します。

(1) リソースアダプタの種類

アプリケーション設定操作で管理する，J2EE アプリケーションに含めて使用するリソースアダプタの種類は次のとおりです。

- DB Connector
  JDBC インタフェースを使用してデータベースに接続する場合に使用します。
  グローバルトランザクションおよびクラスタ構成の場合は利用できません。
- TP1 Connector
  OpenTP1 の SPP に接続する場合に使用します。
  グローバルトランザクションの場合は利用できません。
- そのほかのリソースアダプタ（Connector 1.5 に準拠したリソースアダプタ）
  任意のリソースに接続するためのリソースアダプタとして，Connector 1.5 に準拠したリソースアダプタを使用できます。
  J2EE アプリケーションに含めて利用できるリソースアダプタの種類については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.2 リソースアダプタの種類」を参照してください。

(2) リソースアダプタの管理の流れ

J2EE アプリケーションに含まれるリソースアダプタを管理する基本的な流れを次に示します。

1. J2EE アプリケーションへのリソースアダプタの追加
2. J2EE アプリケーションに含まれるリソースアダプタのプロパティ定義
3. J2EE アプリケーションに含まれるリソースアダプタの接続テスト
4. J2EE アプリケーションの開始・停止
5. J2EE アプリケーションのエクスポート

各手順について説明します。

- **J2EE アプリケーションへのリソースアダプタの追加**

  次のどちらかの方法で，J2EE アプリケーションにリソースアダプタを追加します。

  - アプリケーション開発環境で，リソースアダプタを追加した J2EE アプリケーションを作成します。その J2EE アプリケーションを J2EE サーバにインポートします。
  - リソースアダプタを J2EE サーバにインポートして，J2EE アプリケーションに追加します。

- **J2EE アプリケーションに含まれるリソースアダプタのプロパティ定義**

  データベースと接続するための情報を定義します。

- **J2EE アプリケーションに含まれるリソースアダプタの接続テスト**

  接続テストを実施して，J2EE アプリケーションに含まれるリソースアダプタが正しく動くことを確認します。

- **J2EE アプリケーションの開始・停止**

  設定が完了したリソースアダプタを含む J2EE アプリケーションを開始します。J2EE アプリケーションの開始で，J2EE アプリケーションに含まれるリソースアダプタが開始されます。

  また，運用に応じて J2EE アプリケーションを停止します。J2EE アプリケーションの停止で，J2EE アプリケーションに含まれるリソースアダプタが停止されます。

- **J2EE アプリケーションのエクスポート**

  運用に応じて，リソースアダプタを J2EE アプリケーションに含めてエクスポートします。

  データベースと接続する場合，データベースの設定が必要になります。

  また，リソースアダプタを使用する場合，J2EE アプリケーションでリソースアダプタのリファレンスを解決する必要があります。リソースアダプタのリファレンス解決は，J2EE アプリケーションのカスタマイズで実行します。J2EE アプリケーションのカスタマイズの概要については，「1.3　J2EE アプリケーションの管理」を参照してください。

(3)　リソースアダプタのアプリケーション設定操作

J2EE アプリケーションに含まれるリソースアダプタの管理で使用する，アプリケーション設定操作の機能を次に示します。

表 1-3　J2EE アプリケーションに含まれるリソースアダプタの管理に使用するアプリケーション設定操作

J2EE リソース管理の操作	機能	コマンド
リソースアダプタのインポート	リソースアダプタを J2EE サーバにインポートします。	cjimportres（-type rar 指定）
J2EE アプリケーションへのリソースアダプタの追加	J2EE アプリケーションに，リソースアダプタを追加します。	cjaddapp（-type rar 指定）

J2EE リソース管理の操作	機能	コマンド
J2EE アプリケーションのインポート	リソースアダプタを含む J2EE アプリケーションをインポートします。	cjimportapp
リソースアダプタのプロパティ定義	J2EE アプリケーションに含まれるリソースアダプタの属性を取得して，属性ファイルを生成します。	cjgetappprop（-type rar 指定）
	J2EE アプリケーションに含まれるリソースアダプタの属性を，属性ファイルに指定された値に変更します。	cjsetappprop（-type rar 指定）
リソースアダプタの接続テスト	J2EE アプリケーションに含まれるリソースアダプタの接続テストをします。	cjtestres（-name ［アプリケーション名］ -type rar 指定）
J2EE アプリケーションの開始と停止	J2EE アプリケーションを開始します。J2EE アプリケーションの開始で，J2EE アプリケーションに含まれるすべてのリソースアダプタが開始されます。リソースアダプタ単位の開始はできません。	cjstartapp
	開始されている J2EE アプリケーションを停止します。J2EE アプリケーションの停止で，J2EE アプリケーションに含まれるすべてのリソースアダプタが停止されます。リソースアダプタ単位の停止はできません。	cjstopapp
J2EE アプリケーションのエクスポート	J2EE アプリケーションに含めて，リソースアダプタをエクスポートします。	cjexportapp
J2EE アプリケーションからリソースアダプタの削除	J2EE アプリケーションから，J2EE リソースのリソースアダプタを削除します。	cjdeleteapp（-type rar 指定）
リソースの一覧表示	J2EE アプリケーションに含まれるリソースアダプタの一覧を表示します。	cjlistapp（-type rar 指定）
そのほかの操作	J2EE アプリケーションに含まれるリソースアダプタのコネクションプールの状態を表示します。	cjlistpool（-name ［アプリケーション名］指定）
	J2EE アプリケーションに含まれるリソースアダプタのコネクションプールのコネクションを削除します。	cjclearpool（-name ［アプリケーション名］指定）

1.2.4　リソースアダプタ以外の J2EE リソースのアプリケーション設定操作

リソースアダプタ以外の J2EE リソースの，アプリケーション設定操作について説明します。

(1)　リソースアダプタ以外の J2EE リソースの種類

アプリケーション設定操作で管理する，リソースアダプタ以外の J2EE リソースは次のとおりです。

- JavaBeans リソース
- メールコンフィグレーション

(2) J2EE リソース管理の流れ

J2EE リソースの管理の基本的な流れを次に示します。

1. J2EE リソースのインポート
2. J2EE リソースのプロパティ定義
3. J2EE リソースの接続テスト
4. J2EE リソースの開始・停止

サーバ管理コマンドを使用した J2EE リソースの管理では，J2EE リソースの種別によって，必要な設定および操作方法が異なります。

J2EE リソースの種別ごとに，それぞれの手順で必要な作業を次の表に示します。手順の列に記載されている番号に従った順序で作業を実施してください。

表 1-4 J2EE リソースの管理で行う必要な作業

手順		JavaBeans リソース	メールコンフィグレーション
1.	J2EE リソースのインポート	○	－
2.	J2EE リソースのプロパティ定義	○	○※
3.	J2EE リソースの接続テスト	－	○
4.	J2EE リソースの開始・停止	○	－

（凡例） ○：必要 －：該当項目なし

注※ メールコンフィグレーションのプロパティ定義には，メールコンフィグレーションの新規作成が含まれます。

各手順について説明します。

- **J2EE リソースのインポート**
  JavaBeans リソースをインポートします。
- **J2EE リソースのプロパティ定義**
  JavaBeans リソースやメールコンフィグレーションを管理する情報を定義します。
- **J2EE リソースの接続テスト**
  メールコンフィグレーションが正しく動くことを確認します。
- **J2EE リソースの開始・停止**
  設定が完了した JavaBeans リソースを開始します。また，運用に応じて JavaBeans リソースを停止します。

(3) J2EE リソースのアプリケーション設定操作

J2EE リソース管理で使用する，アプリケーション設定操作の機能を次に示します。

表 1-5 J2EE リソースの管理に使用するアプリケーション設定操作

J2EE リソース管理の操作	機能	コマンド
J2EE リソースのインポート	JavaBeans リソースを J2EE サーバにインポートします。	cjimportjb
J2EE リソースのプロパティ定義	JavaBeans リソースの属性を取得して，属性ファイルを生成します。	cjgetjbprop
	JavaBeans リソースの属性を，属性ファイルに指定された値に変更します。	cjsetjbprop
	メールに含まれるリソースの属性を取得して，属性ファイルを生成します。	cjgetresprop（-type mail 指定）
	メールに含まれるリソースの属性を，属性ファイルに指定された値に変更します。	cjsetresprop（-type mail 指定）
J2EE リソースの接続テスト	メールの接続テストをします。	cjtestres（-type mail 指定）
J2EE リソースの開始と停止	JavaBeans リソースを開始します。	cjstartjb
	JavaBeans リソースを停止します。	cjstopjb
リソースの一覧表示	インポート済みの JavaBeans リソースの一覧を表示します。	cjlistjb
	インポート済みのメール一覧を表示します。	cjlistres（-type mail 指定）
リソースの削除	JavaBeans リソースを削除します。	cjdeletejb
	メールを削除します。	cjdeleteres（-type mail 指定）
そのほかの操作	メールの属性をコピーします。	cjcopyres（-type mail 指定）

1.3　J2EE アプリケーションの管理

この節では，アプリケーション設定操作での J2EE アプリケーションの管理の概要について説明します。

1.3.1　管理する J2EE アプリケーション

管理する J2EE アプリケーションの形式と，J2EE アプリケーションを構成するコンポーネントについて説明します。

(1)　J2EE アプリケーションの形式

アプリケーション設定操作で管理するのは，次のどちらかの形式のアプリケーションです。

- **アーカイブ形式の J2EE アプリケーション**
  J2EE サーバの作業ディレクトリ下に，J2EE アプリケーションのコンポーネントの構成ファイルを持つ形式です。
- **展開ディレクトリ形式の J2EE アプリケーション**
  J2EE サーバの作業ディレクトリ下に，J2EE アプリケーションのコンポーネントの構成ファイルを持たないで，J2EE サーバの外部にある構成ファイルを論理的な J2EE アプリケーションとして使用する形式です。

J2EE アプリケーションの形式の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「18.　J2EE アプリケーションの形式とデプロイ」を参照してください。

(2)　J2EE アプリケーションを構成するコンポーネント

アプリケーション設定操作では，次のコンポーネントで構成される J2EE アプリケーションを管理します。

- **サーブレット・JSP で構成された J2EE アプリケーション**
- **Enterprise Bean で構成された J2EE アプリケーション**
- **サーブレット・JSP と Enterprise Bean で構成された J2EE アプリケーション**

また，J2EE アプリケーションのコンポーネントにリソースアダプタを含めて，管理することもできます。

アーカイブ形式の J2EE アプリケーションの場合は，コンポーネントは，J2EE アプリケーションへの追加や新規 J2EE アプリケーションの作成で使うことができます。J2EE アプリケーションの作成については，「7.　J2EE アプリケーションの作成」を参照してください。

1.3.2　J2EE アプリケーション管理の流れ

J2EE アプリケーションの管理の流れを次に示します。

1. ファイルのインポート

2. J2EE アプリケーションの作成※

3. J2EE アプリケーションのプロパティ定義

4. J2EE アプリケーションの開始・停止

5. J2EE アプリケーションのエクスポート

注※　すでにアプリケーション開発ツールなどを使用して作成，設定されている J2EE アプリケーションの場合は実施しません。

J2EE アプリケーションの管理で必要な作業を次の表に示します。手順の列に記載されている番号に従った順序で作業を実施してください。

表 1-6　J2EE アプリケーションの管理で行う必要な作業

手順		新規に作成する J2EE アプリケーション	作成済みの J2EE アプリケーション※1	
			アーカイブ形式	展開ディレクトリ形式
1.	ファイルのインポート	○※2	○	○
2.	J2EE アプリケーションの作成	○	―※3	―
3.	J2EE アプリケーションのプロパティ定義	○	○	○※4
4.	J2EE アプリケーションの開始・停止	○	○	○
5.	J2EE アプリケーションのエクスポート	○	○	○

（凡例）　○：必要　―：該当項目なし

注※1　アプリケーション開発ツールなどを使用して作成した J2EE アプリケーションです。J2EE アプリケーションの形式によって，インポート方法が異なります。

- アーカイブ形式の J2EE アプリケーションを設定する場合，アーカイブファイルのパスを指定して J2EE アプリケーションをインポートしてください。
- 展開ディレクトリ形式の J2EE アプリケーションを設定する場合，アプリケーションディレクトリパスまたは展開ディレクトリパスを指定して，J2EE アプリケーションをインポートしてください。

注※2　コンポーネントの構成ごとに，次のファイルをインポートします。

- サーブレット・JSP が含まれる J2EE アプリケーションを作成する場合，サーブレット・JSP をインポートしてください。
- Enterprise Bean が含まれる J2EE アプリケーションを作成する場合，Enterprise Bean をインポートしてください。
- リソースアダプタが含まれる J2EE アプリケーションを作成する場合，リソースアダプタをインポートしてください。

注※3　作成済みのアーカイブ形式の J2EE アプリケーションに，コンポーネントやライブラリ JAR を追加・削除できます。

注※4　サーバ管理コマンドを実行するホストと異なるホストで稼働している J2EE サーバの展開ディレクトリ形式の J2EE アプリケーションについては，J2EE アプリケーションの設定およびカスタマイズはできません。

各手順について説明します。

- **ファイルのインポート**
  J2EE アプリケーションを構成するコンポーネント（サーブレット・JSP，Enterprise Bean，リソースアダプタおよびライブラリ JAR）や，作成済みの J2EE アプリケーションをインポートします。
- **J2EE アプリケーションの作成**
  インポートしたファイルで，J2EE アプリケーションを作成します。
- **J2EE アプリケーションのプロパティ定義**
  作成した J2EE アプリケーションまたはインポートした作成済みの J2EE アプリケーションに対して，J2EE アプリケーションを構成するコンポーネントに応じた属性を設定します。
  - DD 文に対応する属性の定義
  - 実行時プロパティや動作の設定
  - J2EE リソースのリファレンス解決
- **J2EE アプリケーションの開始・停止**
  設定やカスタマイズの完了した J2EE アプリケーションを開始します。また，運用に応じて停止します。
- **J2EE アプリケーションのエクスポート**
  運用に応じて，J2EE アプリケーションをエクスポートします。

1.3.3 J2EE アプリケーション管理に使用するアプリケーション設定操作

J2EE アプリケーション管理で使用する，アプリケーション設定操作の機能を次に示します。

表 1-7 J2EE アプリケーションの管理に使用するサーバ管理コマンド

J2EE アプリケーション管理の操作	機能	コマンド
ファイルのインポート	J2EE アプリケーションを構成するコンポーネント（WAR ファイル，EJB-JAR ファイル，RAR ファイル）を，J2EE サーバにインポートします。	cjimportres
	ライブラリ JAR ファイルを，J2EE サーバにインポートします。	cjimportlibjar
	J2EE アプリケーションを，J2EE サーバにインポートします。	cjimportapp
J2EE アプリケーションの作成	インポート済みの EJB-JAR ファイル，WAR ファイル，RAR ファイルを J2EE アプリケーションに追加します。	cjaddapp
J2EE アプリケーションのプロパティ定義	J2EE アプリケーションまたは J2EE アプリケーションに含まれるコンポーネントの属性を取得して，属性ファイルを生成します。	cjgetappprop
	J2EE アプリケーションまたは J2EE アプリケーションに含まれるコンポーネントの属性を，指定されたアプリケーション属性ファイルの値に変更します。	cjsetappprop

J2EE アプリケーション管理の操作	機能	コマンド
J2EE アプリケーションの開始と停止	J2EE アプリケーションを開始して，クライアントからのリクエストを受け取ることができるようにします。	cjstartapp
	J2EE アプリケーションを停止して，クライアントからのリクエストを受け取らないようにします。	cjstopapp
J2EE アプリケーションのエクスポート	J2EE サーバ上の J2EE アプリケーションをエクスポートします。	cjexportapp
J2EE アプリケーションの一覧表示	すべての J2EE アプリケーションについての名称と状態，J2EE アプリケーションに含まれる EJB-JAR ファイル，WAR ファイル，または RAR ファイルの一覧を表示します。	cjlistapp
	J2EE アプリケーションに含まれるライブラリ JAR の一覧を表示します。	cjlistlibjar
	J2EE サーバで稼働しているトランザクション情報の一覧を表示します。	cjlisttrn
	J2EE サーバで停止中のトランザクション情報の一覧を表示します。	cjlisttrnfile
J2EE アプリケーションの削除	J2EE アプリケーションまたは J2EE アプリケーションに含まれる EJB-JAR ファイル，WAR ファイル，RAR ファイルを，指定された J2EE サーバから削除します。	cjdeleteapp
	ライブラリ JAR を J2EE アプリケーションから削除します。	cjdeletelibjar
そのほかの操作	展開ディレクトリ形式の J2EE アプリケーションを入れ替えます。※	cjreloadapp
	アーカイブ形式の J2EE アプリケーションを入れ替えます。※	cjreplaceapp
	J2EE アプリケーションの名称を変更します。	cjrenameapp
	起動されているリソースアダプタのコネクションプールの状態を表示します。	cjlistpool
	CMP2.x Entity Bean 用の SQL 文を生成します。	cjgencmpsql
	ロールにユーザを追加します。	cjmapsec
	ユーザまたはロールを追加します。	cjaddsec
	ユーザまたはロールを削除します。	cjdeletesec
	ユーザまたはロールの一覧を表示します。	cjlistsec
	J2EE アプリケーションについて，RMI-IIOP スタブおよびインタフェースを取得します。	cjgetstubsjar

注※　J2EE アプリケーションの形式については，「1.3.1　管理する J2EE アプリケーション」を参照してください。

1.4　アプリケーション設定操作の制約

この節では，J2EE アプリケーション設定操作を実行する場合の，実行時ホストおよび J2EE アプリケーションの状態による制約について説明します。

1.4.1　アプリケーション設定操作の実行ホストについての制約

アプリケーション設定操作は，J2EE サーバおよび CORBA ネーミングサービスとは別のホストから実行できます。

ただし，展開ディレクトリ形式の J2EE アプリケーションの場合，J2EE アプリケーションの設定とカスタマイズについては J2EE サーバと同じホストだけで実行できます。

別のホストの J2EE サーバを操作する場合，あらかじめ次の設定が必要です。

- J2EE サーバへのアクセス権の設定

  操作対象の J2EE サーバへのアクセス権限が必要です。J2EE サーバへのアクセス権を設定するには，J2EE サーバ用の usrconf.properties ファイルで，webserver.connector.http.permitted.hosts キーを定義します。J2EE サーバ用の usrconf.properties ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。

  J2EE サーバに対するアクセス権限を持っていないときには，エラーメッセージが出力され，アプリケーション設定操作は実行できません。

- ネットワークの設定

  サーバ管理コマンド実行ホストから J2EE サーバ実行ホストのホスト名称を解決できるよう，あらかじめネットワークを設定しておく必要があります。

アプリケーション設定操作，J2EE サーバおよび CORBA ネーミングサービスを別のホストで実行する場合，それぞれにインストールされているアプリケーションサーバのバージョンが異なると正常に実行できないときがあります。それぞれに同じバージョンのアプリケーションサーバがインストールされている環境で使用してください。

1.4.2　J2EE アプリケーションの状態による操作の制約

J2EE アプリケーションの操作には，J2EE アプリケーションの状態によって実行できるものと実行できないものがあります。

J2EE アプリケーションの状態を次の表に示します。

表 1-8　J2EE アプリケーションの状態

項番	状態※	説明
1	開始	J2EE アプリケーションが開始されています。
2	閉塞中	閉塞処理が実施されています。
3	閉塞	閉塞処理が正常終了しました。
4	通常停止中	通常停止処理が実施されています。
5	強制停止中	強制停止操作で強制停止処理が実施されています。
6	停止	J2EE アプリケーションが停止されています。
7	閉塞失敗	閉塞処理が異常終了しました。
8	通常停止失敗	通常停止処理が異常終了しました。
9	強制停止失敗	強制停止処理が異常終了しました。

注※　J2EE アプリケーションの状態を参照する方法については「10.3　J2EE アプリケーションの一覧の参照」を参照してください。

J2EE アプリケーションがそれぞれの状態の場合に実行できる操作，および実行できない操作を次の表に示します。

表 1-9　J2EE アプリケーションの状態と実行できる操作

操作（コマンド）		J2EE アプリケーションの状態								
		1	2	3	4	5	6	7	8	9
ファイルのインポート	WAR ファイル，EJB-JAR ファイルおよび RAR ファイルのインポート（cjimportres）	○	○	○	○	○	○	○	○	○
	ライブラリ JAR ファイルのインポート（cjimportlibjar）	－	－	－	－	－	○	－	－	－
J2EE アプリケーションの作成（cjaddapp）		－	－	－	－	－	○	－	－	－
J2EE アプリケーションの設定とカスタマイズ	属性ファイルの取得（cjgetappprop）※1	○	－	－	－	－	○	－	－	－
	属性の設定（cjsetappprop）※1	－	－	－	－	－	○	－	－	－
J2EE アプリケーションの開始と停止	J2EE アプリケーションの開始（cjstartapp）	－	－	－	－	－	○	－	－	－
	J2EE アプリケーションの通常停止（cjstopapp）	○	－	－	－	－	－	－	－	－
	J2EE アプリケーションの手動強制停止（cjstopapp）	－	－	○	○	－	－	－	－	－

操作（コマンド）		J2EE アプリケーションの状態								
		1	2	3	4	5	6	7	8	9
	J2EE アプリケーションの自動強制停止（cjstopapp）	○	−	−	−	−	−	−	−	−
J2EE アプリケーションのエクスポート（cjexportapp）		○	−	−	−	−	○	−	−	−
J2EE アプリケーションの一覧表示	J2EE アプリケーションの状態表示（cjlistapp）	○	○	○	○	○	○	○	○	○
	ライブラリ JAR の一覧表示（cjlistlibjar）	○	○	○	○	○	○	○	○	○
	稼働中のトランザクション情報一覧表示（cjlisttrn）	○	○	○	○	○	○	○	○	○
	停止中のトランザクション情報一覧表示（cjlisttrnfile）	○	○	○	○	○	○	○	○	○
J2EE アプリケーションの削除	J2EE アプリケーションおよび構成コンポーネントの削除（cjdeleteapp）	−	−	−	−	−	○	−	−	−
	ライブラリ JAR の削除（cjdeletelibjar）	−	−	−	−	−	○	−	−	−
そのほかの操作	展開ディレクトリ形式の J2EE アプリケーションの入れ替え（cjreloadapp）	○	−	−	−	−	○	−	−	−
	アーカイブ形式の J2EE アプリケーションの入れ替え（cjreplaceapp）	○	−	−	−	−	○	−	−	−
	J2EE アプリケーション名称の変更（cjrenameapp）	−	−	−	−	−	○	−	−	−
	リソースアダプタのコネクションプールの状態表示（cjlistpool）	○	−	−	−	−	−	−	−	−
	SQL 文の生成（cjgencmpsql）	−	−	−	−	−	○	−	−	−
	ロールにユーザを登録（cjmapsec）	○	○	○	○	○	○	○	○	○
	ユーザまたはロールの追加（cjaddsec）	○	○	○	○	○	○	○	○	○
	ユーザまたはロールの削除（cjdeletesec）	○	○	○	○	○	○	○	○	○
	ユーザまたはロールの一覧表示（cjlistsec）	○	○	○	○	○	○	○	○	○

操作（コマンド）		J2EE アプリケーションの状態								
		1	2	3	4	5	6	7	8	9
	RMI-IIOP スタブおよびインタフェースの取得（cjgetstubsjar）	○	○	○	○	○	○※2	○	○	○

（凡例）

数字：表 1-8 の項番に対応した，J2EE アプリケーションの状態

○：実行できる

－：実行できない

注※1　J2EE アプリケーションのカスタマイズおよび各コンポーネントのプロパティを変更する場合，J2EE アプリケーションが開始されていても属性ファイルは取得できます。変更内容に編集した属性ファイルの値を反映するためには，J2EE アプリケーションを停止させてください。

注※2　J2EE アプリケーションが一度も開始されていない場合は，実行できません。

1.4.3　そのほかの制約

サーバ管理コマンドによる処理が実行されているときに別のサーバ管理コマンドを実行した場合，処理の実行はサーバ単位および処理を実行しているサーバ管理コマンドの系統ごとに制御されます。サーバ管理コマンドを同時に実行した場合の制御については，「3.2　サーバ管理コマンドの排他制御」を参照してください。

2 アプリケーション設定操作で使用するインタフェース

この章では，アプリケーション設定操作のインタフェースである，サーバ管理コマンドの概要について説明します。

2.1　サーバ管理コマンドの機能

アプリケーション設定操作では，インタフェースとしてサーバ管理コマンドを提供しています。サーバ管理コマンドは，J2EE サーバで動作するリソースや J2EE アプリケーションを設定，および管理するためのコマンド群です。

アプリケーション設定操作で使用するサーバ管理コマンドの機能を次の表に示します。

表 2-1　サーバ管理コマンドの機能一覧

機能		参照先
リソースアダプタの設定	データベースと接続するための設定	4.2
	そのほかのリソースアダプタと接続するための設定	4.3
J2EE アプリケーションに含まれるリソースアダプタの設定		5 章
リソースアダプタ以外の J2EE リソースの設定	JavaBeans リソースの設定	6.2
	メールコンフィグレーションの設定	6.3
J2EE アプリケーション（EAR）の作成		7 章
J2EE アプリケーションのインポート※1 とエクスポート		8 章
J2EE アプリケーションのプロパティの設定		9 章
J2EE アプリケーションの実行	J2EE アプリケーションの開始と停止	10.2
	J2EE アプリケーションの一覧の参照	10.3
	J2EE アプリケーションの削除	10.4
	J2EE アプリケーションの入れ替え※2	10.5
	J2EE アプリケーション名の変更	10.6
	RMI-IIOP スタブとインタフェースの取得	10.7
	トランザクション一覧の参照	10.8

注※1

次の形式の J2EE アプリケーションをインポートできます。

- アーカイブ形式の J2EE アプリケーション（EAR ファイル）
- 展開ディレクトリ形式の J2EE アプリケーション（アプリケーションディレクトリ）

注※2

インポート時の J2EE アプリケーションの形式によって，デプロイした J2EE アプリケーションを入れ替える場合，次のように操作が異なります。

- デプロイしたアーカイブ形式の J2EE アプリケーションを入れ替える場合，一度 J2EE アプリケーションを再起動（停止，入れ替え，開始）する必要があります。

- 展開ディレクトリ形式の J2EE アプリケーションを入れ替える場合，開始中の J2EE アプリケーションを再起動（停止，入れ替え，開始）しないで，サーバ管理コマンドを実行して，アプリケーションを入れ替えることができます（リロード機能）。

アプリケーションの入れ替えについては，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「5.6　J2EE アプリケーションの入れ替え」を参照してください。

3 サーバ管理コマンドの基本操作

この章では，サーバ管理コマンドでアプリケーションを設定する場合の基本的な操作について説明します。

3.1　サーバ管理コマンドの実行の前提条件

サーバ管理コマンドを使用する場合，次のことを確認してください。

- サーバ管理コマンドは，J2EE サーバが起動している状態で実行できるコマンドです。使用する場合は，J2EE サーバおよびその前提プロセスを起動してから実行してください。J2EE サーバおよびその前提プロセスの起動については，マニュアル「アプリケーションサーバ 機能解説 運用／監視／連携編」の「2.3　論理サーバの起動・停止の仕組み」を参照してください。
- サーバ管理コマンドの実行には，root 権限，または Component Container 管理者の権限が必要です。実行に必要な権限を設定してください。
- Windows でアプリケーションサーバのコマンドを使用する場合,管理者特権で実行する必要があります。Windows でアプリケーションサーバのコマンドを使用する場合の注意事項については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「1.4　コマンド使用時の注意事項」を参照してください。

サーバ管理コマンドを使用する場合の注意事項については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「1.4　コマンド使用時の注意事項」を参照してください。

3.2　サーバ管理コマンドの排他制御

サーバ管理コマンドの系統とサーバ管理コマンドの排他制御について説明します。

3.2.1　サーバ管理コマンドの系統

サーバ管理コマンドによる処理が実行されている場合に別のサーバ管理コマンドを実行したとき，処理の実行はサーバ単位および処理を実行しているサーバ管理コマンドの系統ごとに制御されます。

サーバ管理コマンドには次の三つの系統があります。

- **参照系コマンド**
  インポートしたJ2EEアプリケーションの数や状態，含まれるリソース名など，J2EEサーバの構成状態を表示するサーバ管理コマンドです。J2EEサーバの内容は更新しません。
- **更新系コマンド**
  J2EEサーバの内容を更新したり，構成情報を取得したりするサーバ管理コマンドです。
- **特権系コマンド**
  J2EEサーバの内容を更新するコマンドで，常にほかのコマンドよりも優先して処理が実施されるサーバ管理コマンドです。

なお，それぞれのサーバ管理コマンドの系統については，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

3.2.2　サーバ管理コマンドの排他制御

コマンド実行の排他制御の条件を，サーバ管理コマンドの系統ごとに次の表に示します。

表 3-1　コマンド実行の排他制御の条件

実行中のコマンド	あとから実行するコマンド		
	更新系コマンド	参照系コマンド	特権系コマンド
更新系コマンド	○	◎	◎
参照系コマンド	◎	○	◎
特権系コマンド	◎	◎	×

（凡例）
◎：コマンドの実行が許可される。
○：あとから実行するコマンドは実行中のコマンドが終了するまで待つ。
×：コマンドの実行を中止する。排他エラーが返る。

ほかのコマンドの実行中に別のサーバ管理コマンドを実行しようとした場合，実行するコマンドの種類によって，異なる形で制御されます。

(1) サーバ管理コマンドが処理をしている J2EE サーバに対して異なるサーバ管理コマンドを実行した場合の制御

サーバ管理コマンドが処理をしている J2EE サーバに対して異なる系統のサーバ管理コマンドを実行した場合，それぞれのサーバ管理コマンドの処理は並行して実行されます。また，サーバ管理コマンドが処理をしている J2EE サーバに対して，同じ系統のサーバ管理コマンドを同時に実行した場合，あとに実行したサーバ管理コマンドの処理要求は，実行された順番で待ち状態になり，先に実行したサーバ管理コマンドの処理が終了してから開始されます。したがって，同時に処理を実行できるサーバ管理コマンドの数は，それぞれの系統のサーバ管理コマンドで一つずつです。また，待ち状態にあるサーバ管理コマンドには最大数の制限はありません。処理の実行が待ち状態になると，J2EE サーバ側にメッセージ KDJE42211-I が表示されます。

(2) シャットダウンコマンドの処理中にサーバ管理コマンドを使用する場合の制御

シャットダウンコマンドが処理中，または実行待ち状態にある J2EE サーバに対しては，更新系と参照系に分類されるサーバ管理コマンドは実行できません。シャットダウン処理時に，参照系と更新系に分類されるサーバ管理コマンドを実行した場合は排他エラーとなり，メッセージ KDJE42212-E が表示されます。なお，特権系に分類されるサーバ管理コマンドは実行できます。

(3) 特権系コマンドを使用する場合の制御

特権系コマンドは，ほかのすべてのコマンドより優先して実行できます。また，特権系コマンドの実行中であっても，ほかのコマンドの実行に対して影響することはありません。ただし，一つの J2EE サーバに対して，同時に実行できる特権系コマンドは一つです。二つ以上の特権系コマンドを一つの J2EE サーバに実行した場合はエラーとなり，メッセージ KDJE42230-E が表示されます。

3.2.3 サーバ管理コマンドの排他制御の強制解除

しばらく時間を置いて，何度かサーバ管理コマンドを実行しても同じ排他エラーメッセージ（実行中コマンドの情報も同じ）が表示される場合，サーバ管理コマンドの異常終了などによって，サーバ管理コマンドの排他情報に矛盾が生じているおそれがあります。この場合，メッセージ中に示されている実行中コマンドが，実際には実行されていないことをよく確認した上で，サーバ管理コマンドの排他制御を強制解除し，排他情報をリセットしてください。

排他制御の強制解除を実施するには，強制解除の対象にする J2EE サーバを指定する必要があります。

注意

サーバ管理コマンド実行中に排他制御を解除すると，複数のサーバ管理コマンドが同時に実行され，J2EE サーバに矛盾が生じることがあります。このため，これらのサーバ管理コマンドが正常に実行されているときに誤って排他情報を解除しないように注意してください。

サーバ管理コマンドの排他制御を強制解除する手順を次に示します。

1. **排他制御を強制解除する前に，サーバ管理コマンドが実際には実行中でないことを確認します。**

   サーバ管理コマンドを実行しようとすると，メッセージ KDJE37057-E または KDJE37301-E が表示されます。これらのメッセージに示されるサーバ管理コマンドが実際には実行されていないことをよく確認してください。

2. **cjresetsv コマンドを実行して，排他制御を強制解除します。**

   実行形式

   ```
   cjresetsv ［＜J2EEサーバ名＞］ ［-nameserver ＜プロバイダURL＞］
   ```

   実行例

   cjresetsv MyServer

3.3　サーバ管理コマンドの動作設定のカスタマイズ

ユーザ定義ファイルを編集すると，サーバ管理コマンドの動作設定をカスタマイズできます。必要に応じて，サーバ管理コマンドの動作設定をカスタマイズしてください。なお，UNIX の場合，カスタマイズには root 権限，または Component Container 管理者の権限が必要です。

次のディレクトリにあるコマンドがサーバ管理コマンドのコマンドになります。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥bin
- UNIX の場合
  /opt/Cosminexus/CC/admin/bin

コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

3.3.1　カスタマイズ対象のユーザ定義ファイル

サーバ管理コマンドの動作設定をカスタマイズするには，次のユーザ定義ファイルをテキストエディタなどで編集します。

- usrconf.properties（サーバ管理コマンド用システムプロパティファイル）
  サーバ管理コマンドのプロパティを指定するファイルです。
  ファイルの格納場所を次に示します。
  - Windows の場合
    <Application Server のインストールディレクトリ>¥CC¥admin¥usrconf¥usrconf.properties
  - UNIX の場合
    /opt/Cosminexus/CC/admin/usrconf/usrconf.properties

  詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「5.2.3 usrconf.properties（サーバ管理コマンド用システムプロパティファイル）」を参照してください。
- usrconf.bat（サーバ管理コマンド用オプション定義ファイル）
  Windows の場合にサーバ管理コマンドの JavaVM の起動オプションを指定するファイルです。
  ファイルの格納場所を次に示します。
  <Application Server のインストールディレクトリ>¥CC¥admin¥usrconf¥usrconf.bat
  詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「5.2.2 usrconf.bat（サーバ管理コマンド用オプション定義ファイル）」を参照してください。
- usrconf（サーバ管理コマンド用オプション定義ファイル）
  UNIX の場合にサーバ管理コマンドの JavaVM の起動オプションを指定するファイルです。

ファイルの格納場所を次に示します。

/opt/Cosminexus/CC/admin/usrconf/usrconf

詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「5.2.1 usrconf（サーバ管理コマンド用オプション定義ファイル)」を参照してください。

3.3.2　サーバ管理コマンドのカスタマイズで設定できる主な項目

サーバ管理コマンドのカスタマイズでユーザ定義ファイルを編集して設定できる項目のうち，主な項目とその項目を設定するユーザ定義ファイルおよびキーを次の表に示します。キーの詳細，およびここで説明していないキーについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「5.　サーバ管理コマンドで使用するファイル」を参照してください。

表 3-2　サーバ管理コマンドのカスタマイズで設定できる主な項目

分類	項目	設定ファイル	設定方法
JavaVM	JavaVM のヒープサイズ	usrconf.bat（Windows の場合)，または usrconf（UNIX の場合）	add.jvm.arg キーに，JavaVM のヒープサイズを指定します。
ログ	TPBroker のトレースファイル	usrconf.properties	vbroker.orb.htc.tracePath キーに，TPBroker のトレースファイル出力先のパスを指定します。
	サーバ管理コマンドのログ	• usrconf.bat（Windows の場合)，または usrconf（UNIX の場合） • usrconf.properties	サーバ管理コマンドのログは，ログの出力先，ログレベルを変更できます。 詳細は，「3.4　サーバ管理コマンドのログ取得の設定」を参照してください。

（凡例）－：該当しません。

注　ネーミング管理の機能を使用する場合にもサーバ管理コマンドのカスタマイズが必要です。詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3.5　実行環境での設定」を参照してください。

3.4　サーバ管理コマンドのログ取得の設定

サーバ管理コマンドのログは，ログの出力先，ログレベルを変更できます。変更できる項目と，項目に対応するユーザ定義ファイルとキーを次に示します。

項目	対応するユーザ定義ファイルとキー
ログの出力先	usrconf.bat（Windows の場合），または usrconf（UNIX の場合）の USRCONF_JVM_ARGS キーの-Dejbserver.log.directory オプション
ログレベル	usrconf.properties の ejbserver.logger.enabled.*キー

ファイルおよびキーの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「5.　サーバ管理コマンドで使用するファイル」を参照してください。

注意事項

サーバ管理コマンドのログ出力ディレクトリに必要なディスク容量は，ejbserver.cui.logfile.compatible キーの指定値によって異なります。

指定値が true の場合

51,655KB ＋ TPBroker のトレース情報

指定値が false の場合

18,624KB ＋ TPBroker のトレース情報

TPBroker のトレース情報を出力するために必要なディスク容量については，マニュアル「TPBroker 運用ガイド」のディスク占有量に関する説明を参照してください。

3.4.1　ログの出力先の変更

サーバ管理コマンドのログの出力先を変更する場合は，サーバ管理コマンド用の usrconf.bat（Windows の場合），または usrconf（UNIX の場合）でログの出力先ディレクトリを指定します。

ログの出力先の変更

デフォルトのログ出力先を次に示します。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥logs
- UNIX の場合
  /opt/Cosminexus/CC/admin/logs/

サーバ管理コマンドのログの出力先は，usrconf.bat（Windows の場合），または usrconf（UNIX の場合）の USRCONF_JVM_ARGS キーの-Dejbserver.log.directory オプションで変更できます。

設定例

- Windows の場合
  set USRCONF_JVM_ARGS="-Dejbserver.log.directory=C:¥CClogs¥admin"
- UNIX の場合
  set USRCONF_JVM_ARGS="-Dejbserver.log.directory=/CClogs/admin"

カレントディレクトリ

ログ出力先を相対パスで指定する場合のカレントディレクトリは，サーバ管理コマンドを実行したときのカレントディレクトリです。

例えば，ユーザ ID が「user1」でユーザのホームディレクトリが「C:¥Documents and Settings¥user1」の場合に，このユーザ ID でログインすると，ログイン直後のカレントディレクトリは「C:¥Documents and Settings¥user1」になります。ここで，サーバ管理コマンドを実行すると，カレントディレクトリは「C:¥Documents and Settings¥user1」になります。

注意事項

- JavaVM の保守情報および GC のログの出力先は，ユーザ定義ファイルで設定しても，usrconf.cfg の add.jvm.arg=-XX:HitachiJavaLog:<ログ出力先>に設定されている値が優先されます。JavaVM の保守情報および GC のログの出力先を設定する場合は，注意してください。
- Management Server リモート管理機能から操作した場合，サーバ管理コマンドのログ出力先は変更できません。

3.4.2 ログレベルの変更

サーバ管理コマンドのログレベルは，ログの重要度を表します。ログレベルには，「Error」，「Warning」，「Information」，「Debug」の四つがあります。ログレベルを設定すると，設定したレベルのログが出力されます。デフォルトでは，Error レベルのログだけが取得されます。通常はデフォルトのまま利用してください。

ログレベルは，サーバ管理コマンド用の usrconf.properties の次のキーで設定します。

ejbserver.logger.enabled.*=<レベル名>

レベル名には，「Error」，「Warning」，「Information」，「Debug」の文字列を一つ，または複数設定します。複数設定する場合には，レベル名の文字列の間をコンマ（,）で区切ります。

記述例

1. ejbserver.logger.enabled.*=Error
2. ejbserver.logger.enabled.*=Error,Warning
3. ejbserver.logger.enabled.*=Error,Warning,Information

4. ejbserver.logger.enabled.*=Error,Warning,Information,Debug

注意事項

- 記述例の 1.，2.，3.，4.の順に，取得できるログの件数が増加していきます。複数のログレベルを設定してログを取得すると，性能が劣化し，ログファイルの面の切り替えが頻繁に起こるようになります。
- レベル名に「Error」，「Warning」，「Information」，「Debug」以外の文字列，または空の値を設定した場合は，KDJE90009-W のメッセージが出力されます。Error レベルのログは取得されます。

3.5　属性ファイルによるプロパティの設定

サーバ管理コマンドでは，属性ファイルを使って，J2EE リソースおよび J2EE アプリケーションのプロパティを設定します。

なお，cosminexus.xml を使用した属性の設定方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16.2　属性の管理」を参照してください。

3.5.1　J2EE リソースのプロパティの設定手順

リソースアダプタとリソースアダプタ以外の J2EE リソースのプロパティ設定手順について説明します。

(1)　リソースアダプタのプロパティの設定手順

J2EE リソースアダプタとしてデプロイして使用するリソースアダプタのプロパティの設定手順を次に示します。

なお，J2EE アプリケーションに含まれるリソースアダプタのプロパティの設定手順については，「3.5.2 J2EE アプリケーションのプロパティの設定手順」を参照してください。

1. 属性ファイルを取得します。

   デプロイする前のリソースアダプタの属性ファイルを取得する場合は，cjgetresprop コマンドを実行します。リソースアダプタをデプロイしたあとで，リソースアダプタの属性ファイルを取得する場合は，cjgetrarprop コマンドを実行します。

   取得する属性ファイルは，Connector 属性ファイルです。

   cjgetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetresprop（リソースの属性の取得)」を参照してください。cjgetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetrarprop（RAR ファイルの属性の取得)」を参照してください。

2. プロパティ設定項目を編集します。

   取得した属性ファイルをテキストエディタで編集します。属性ファイルの編集項目については，各プロパティの設定の説明を参照してください。

   属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1　Connector 属性ファイル」を参照してください。

3. 属性を設定します。

   デプロイする前のリソースアダプタのプロパティを設定する場合は，cjsetresprop コマンドを実行します。デプロイしているリソースアダプタのプロパティを設定する場合は，cjsetrarprop コマンドを実行します。

   cjsetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetresprop（リソースの属性設定)」を参照してください。cjsetrarprop コマンドの詳細

については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetrarprop（RAR属性設定)」を参照してください。

(2)　リソースアダプタ以外の J2EE リソースのプロパティ設定手順

JavaBeans リソースおよびメールコンフィグレーションのプロパティの設定手順を次に示します。

1. 属性ファイルを取得します。

   JavaBeans リソースの属性ファイルを取得する場合は，cjgetjbprop コマンドを実行します。メールコンフィグレーションの属性ファイルを取得する場合は，cjgetresprop コマンドを実行します。

   取得する属性ファイルは，次のとおりです。

   - JavaBeans リソースの場合

     JavaBeans リソース属性ファイル
   - メールコンフィグレーションの場合

     メール属性ファイル

   cjgetjbprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetjbprop（JavaBeans リソースの属性の取得)」を参照してください。cjgetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetresprop（リソースの属性の取得)」を参照してください。

2. プロパティ設定項目を編集します。

   取得した属性ファイルをテキストエディタで編集します。属性ファイルの編集項目については，各プロパティの設定の説明を参照してください。

   属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「2.2.9　Connector 属性の詳細」を参照してください。

3. 属性を設定します。

   JavaBeans リソースのプロパティを設定する場合は，cjsetjbprop コマンドを実行します。メールコンフィグレーションのプロパティを設定する場合は，cjsetresprop コマンドを実行します。

   cjsetjbprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetjbprop（JavaBeans リソースの属性設定)」を参照してください。cjsetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetresprop（リソースの属性設定)」を参照してください。

3.5.2　J2EE アプリケーションのプロパティの設定手順

J2EE アプリケーションのプロパティの設定手順を次に示します。

1. 属性ファイルを取得します。

アプリケーション属性，および J2EE アプリケーションを構成するコンポーネント（サーブレット・JSP，Enterprise Bean，リソースアダプタ）属性の属性ファイルを取得するため，cjgetappprop コマンドを実行します。

取得する属性ファイルは，プロパティを設定する対象となるコンポーネントに応じて異なります。また，そのアプリケーションのコンポーネントの属性情報を一括して編集する場合は，アプリケーション統合属性ファイルを取得します。

cjgetappprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetappprop（アプリケーションの属性の取得)」を参照してください。

2. プロパティ設定項目を編集します。

取得した属性ファイルをテキストエディタで編集します。属性ファイルの編集項目については，各プロパティの設定の説明を参照してください。

属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3. J2EE アプリケーションの設定で使用する属性ファイル」を参照してください。

3. 属性を設定します。

アプリケーション属性，および J2EE アプリケーションを構成するコンポーネント（サーブレット・JSP，Enterprise Bean，リソースアダプタ）属性の属性ファイルを設定するため，cjsetappprop コマンドを実行します。

cjsetappprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetappprop（アプリケーションの属性設定)」を参照してください。

参考

J2EE アプリケーションに J2EE アプリケーションを構成するコンポーネント（サーブレット・JSP，Enterprise Bean，リソースアダプタ）を追加して，新たに J2EE アプリケーションを作成する場合，追加する前のコンポーネントのプロパティ定義は，cjgetresprop コマンドと cjsetresprop コマンドを使用してください。cjgetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetresprop（リソースの属性の取得)」を参照してください。cjsetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetresprop（リソースの属性設定)」を参照してください。

3.6　サーバ管理コマンドで指定するプロバイダ URL

サーバ管理コマンドでは，オプションに＜プロバイダ URL ＞という値を指定できます。

＜プロバイダ URL ＞を省略して，サーバ管理コマンドを実行した場合，usrconf.properties ファイルに設定した値がデフォルトで指定されます。

usrconf.properties ファイルは次の場所に格納されています。

Windows の場合

```
＜Application Serverのインストールディレクトリ＞¥CC¥admin¥usrconf¥usrconf.properties
```

UNIX の場合

```
/opt/Cosminexus/CC/admin/usrconf/usrconf.properties
```

usrconf.properties ファイルに設定した値で実行する場合，および cjlistapp コマンドで，プロバイダ URL を指定して実行する場合の例を次に示します。

usrconf.properties ファイルに設定した値で実行する場合の例

Windows の場合

```
＜Application Serverのインストールディレクトリ＞¥CC¥admin¥bin¥cjlistapp ［＜サーバ名称＞］
```

UNIX の場合

```
/opt/Cosminexus/CC/admin/bin/cjlistapp ［＜サーバ名称＞］
```

プロバイダ URL を指定して実行する場合の例

Windows の場合

```
＜Application Serverのインストールディレクトリ＞¥CC¥admin¥bin¥cjlistapp ［＜サーバ名称＞］ -nameserver ＜プロバイダURL＞
```

UNIX の場合

```
/opt/Cosminexus/CC/admin/bin/cjlistapp ［＜サーバ名称＞］ -nameserver ＜プロバイダURL＞
```

プロバイダ URL は，次の形式で指定します。

実行形式

```
＜プロトコル名＞::＜ホスト名＞:＜ポート番号＞
```

＜プロトコル名＞

CORBA ネーミングサービスのプロトコル名。corbaname 固定です。iioploc または iiopname を指定した場合，corbaname に読み替えられます。

＜ホスト名＞

CORBA ネーミングサービスの稼働しているホスト名。

＜ポート番号＞

CORBA ネーミングサービスの稼働しているポート番号。

プロバイダ URL で指定した CORBA ネーミングサービス上のサーバ管理コマンド排他情報が解除されます。

3.7　このマニュアルでのサーバ管理コマンドの記載方法

このマニュアルでは，サーバ管理コマンドを使用する場合，次のように記載しています。

- 実行形式および実行例では，その操作に主に関連する引数だけを示して説明しています。指定できる引数の詳細など，コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。
- 実行するコマンドについては，コマンド名だけを表示しています。パスを指定してコマンドを実行する場合，コマンドの格納先を含めたパスを指定してください。

  コマンドの格納先は次のとおりです。

  Windows の場合

  ＜Application Serverのインストールディレクトリ＞¥CC¥admin¥bin¥

  UNIX の場合

  /opt/Cosminexus/CC/admin/bin/

4 リソースアダプタの設定

この章では，J2EE リソースアダプタとしてデプロイして使用するリソースアダプタのアプリケーション設定操作について説明します。J2EE リソースアダプタは，J2EE サーバに共有スタンドアロンモジュールとしてデプロイしたリソースアダプタです。

J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作については，「5.　J2EE アプリケーションに含まれるリソースアダプタの設定」を参照してください。

4.1　リソースアダプタの設定の概要

リソースアダプタの設定とは，リソースアダプタを，J2EE アプリケーションから利用できる状態にするための操作です。

J2EE リソースアダプタとしてデプロイして使用できるリソースアダプタの種類とアプリケーション設定操作の概要について説明します。

4.1.1　利用できるリソースアダプタ

J2EE リソースアダプタとして，デプロイして利用できるリソースアダプタを次に示します。

- DB Connector
- DB Connector for Reliable Messaging および Reliable Messaging
- TP1 Connector
- TP1/Message Queue - Access
- そのほかの Connector 1.0 に準拠したリソースアダプタ，または Connector 1.5 に準拠したリソースアダプタ※

注※　そのほかの Connector 1.0 に準拠したリソースアダプタ，または Connector 1.5 に準拠したリソースアダプタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.2　リソースアダプタの種類」を参照してください。

注意事項

J2EE リソースアダプタと J2EE アプリケーションに含まれるリソースアダプタは，J2EE アプリケーションから同時に使用できます。ただし，J2EE リソースアダプタと J2EE アプリケーションに含まれるリソースアダプタを，同じ表示名で同時に使用することはできません。同じ表示名に設定するとエラーとなります。

4.1.2　設定する項目と操作の概要

次のリソースアダプタの設定でのアプリケーション設定操作の概要を示します。

- データベースと接続するための設定
- そのほかのリソースと接続するための設定（リソースアダプタを使用する場合）
- リソースアダプタ共通の設定

データベース上のキューと接続するための設定については，マニュアル「Reliable Messaging」を参照してください。

(1) データベースと接続するための設定

DB Connector を使用してデータベースに接続する場合に必要な作業です。

表 4-1 データベースと接続するための設定の概要

設定項目	内容	参照先
DB Connector のインポート	DB Connector の RAR ファイルをインポートして，J2EE サーバから使用できるリソースに追加します。	4.2.1
DB Connector のプロパティ定義	データベースとの接続についての次の情報を設定します。 • DB Connector の一般情報 • コンフィグレーションプロパティ • 実行時プロパティ • 別名情報	4.2.2
DB Connector のデプロイ	インポートした RAR ファイルを基に DB Connector をデプロイします。DB Connector は，デプロイすると J2EE リソースアダプタとして利用できます。	4.2.3
DB Connector の接続テスト	DB Connector に設定した内容が正しいことを検証します。	4.2.4
DB Connector の開始	DB Connector を開始します。	4.2.5
DB Connector の停止	DB Connector を停止します。	4.2.6
DB Connector のアンデプロイ	デプロイされている DB Connector を削除します。リソースアダプタを入れ替えする場合に，必要に応じて実行してください。	4.2.7
DB Connector のエクスポート	J2EE リソースアダプタを RAR ファイルとしてエクスポートします。 必要に応じて実行してください。	4.2.8

(2) そのほかのリソースと接続するための設定（リソースアダプタを使用する場合）

リソースアダプタを使用して OpenTP1 などの各種リソースに接続する場合に必要な作業です。

表 4-2 そのほかのリソースと接続するための設定（リソースアダプタを使用する場合）の概要

設定項目	内容	参照先
リソースアダプタのインポート	リソースアダプタ（RAR ファイル）をインポートして，J2EE サーバから使用できるリソースに追加します。	4.3.1
リソースアダプタのプロパティ定義	リソースとの接続についての次の情報を設定します。 • リソースアダプタの一般情報 • コンフィグレーションプロパティ	4.3.2 または 4.3.3

設定項目	内容	参照先
	• 実行時プロパティ • 別名情報	
リソースアダプタのデプロイ	インポートしたリソースアダプタを基にリソースアダプタをデプロイします。 リソースアダプタは，デプロイすると J2EE リソースアダプタとして利用できます。	4.3.4
J2EE リソースアダプタの接続テスト	リソースアダプタに設定した内容が正しいかどうかを検証します。	4.3.5
J2EE リソースアダプタの開始	J2EE リソースアダプタを開始します。	4.3.6
J2EE リソースアダプタの停止	J2EE リソースアダプタを停止します。	4.3.7
J2EE リソースアダプタのアンデプロイ	J2EE リソースアダプタを削除します。 リソースアダプタを入れ替えする場合に，必要に応じて実行してください。	4.3.8
J2EE リソースアダプタのエクスポート	J2EE リソースアダプタを RAR ファイルとしてエクスポートします。 必要に応じて実行してください。	4.3.9

(3)　リソースアダプタ共通の設定

リソースアダプタに共通の設定です。

表 4-3　リソースアダプタに共通な設定の概要

設定項目	内容	参照先
J2EE リソースアダプタの状態と一覧の表示	デプロイされている J2EE リソースアダプタの状態，コネクション定義識別子の一覧などを参照します。	4.4
リソースアダプタの一覧の参照	インポートされているリソースアダプタの一覧，コネクション定義識別子の一覧などを参照します。	4.5
コネクションプールの状態の確認	リソースアダプタのコネクションプールの状態を参照します。また，必要に応じてコネクションプール内のコネクションを削除します。	4.6
JNDI 名前空間に登録されるリソースアダプタ名の参照と変更	リソースアダプタの JNDI 名前空間への登録名を参照して，必要に応じて別名を付与します。	4.7
リソースアダプタの削除	リソースアダプタを削除します。	4.8
リソースアダプタのコピー	リソースアダプタをほかのフォルダなどにコピーできます。	4.9

4.2　データベースと接続するための設定

データベースと接続するために，DB Connector の設定をします。DB Connector はデータベースと接続するためのリソースアダプタです。

ここで説明する DB Connector を使用したデータベースへの接続とは，JDBC インタフェースを使用してデータベースのテーブルだけにアクセスすることです。

JMS インタフェースを使用してキューにアクセスする場合は，DB Connector for Reliable Messaging および Reliable Messaging を使用してデータベースに接続します。詳細は，マニュアル「Reliable Messaging」を参照してください。

DB Connector の設定をするには，あらかじめ使用するデータベースの環境を設定しておいてください。また，使用する DB Connector の種類に応じて，HiRDB Type4 JDBC Driver，Oracle JDBC Thin Driver，SQL Server JDBC Driver，MySQL Connector/J，または PostgreSQL JDBC Driver の設定も必要です。

DB Connector の設定は次の手順で実施します。

1. DB Connector をインポートします。
2. プロパティを定義します。
3. DB Connector をデプロイします。
   デプロイとは，DB Connector を J2EE サーバに共有スタンドアロンモジュール（J2EE リソースアダプタ）として配備することです。
4. 接続を確認します。
   正しく接続できるかどうかは，接続テストによって確認できます。

デプロイした DB Connector は，J2EE アプリケーションのプロパティ設定で，リソースアダプタのリファレンスを解決する必要があります。詳細については，「9.3.3　リソースアダプタのリファレンス定義」を参照してください。

参考

DB Connector のプロパティを新規に設定する場合，Component Container が提供しているテンプレートファイルが利用できます。

Connector 属性ファイルのテンプレートファイルは，次に示すディレクトリに格納されています。

- Windows の場合
  <Application Server のインストールディレクトリ>¥CC¥admin¥templates¥
- UNIX の場合

/opt/Cosminexus/CC/admin/templates/

このテンプレートファイルを使用すると，DB Connector をインポートする前に，Connector 属性ファイルを編集しておくことができます。テンプレートファイルはコピーして使用してください。

Connector 属性ファイルのテンプレートファイル名については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.13　Connector 属性ファイルのテンプレートファイル」を参照してください。

なお，すでにプロパティが設定されている DB Connector のプロパティを変更する場合は，テンプレートファイルは使用しないでください。インポートした DB Connector の Connector 属性を取得して，Connector 属性ファイルを編集してください。

4.2.1　DB Connector のインポート

次に示すコマンドを実行して DB Connector をインポートします。

(1)　実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -f ＜ファイルパス＞
```

＜ファイルパス＞には，RAR ファイルを指定してください。

RAR ファイルは，次のディレクトリに格納されています。

- Windows の場合
  ＜Application Serverのインストールディレクトリ＞¥CC¥DBConnector¥
- UNIX の場合
  /opt/Cosminexus/CC/DBConnector/

DB Connector をリソースアダプタとしてインポートする場合，トランザクションの管理方法，および使用する JDBC ドライバの種類によって，次のどれかの RAR ファイルを指定します。なお，XDM/RD E2，SQL Server に接続する場合，グローバルトランザクションは使用できません。

- DBConnector_HiRDB_Type4_CP.rar
  HiRDB Type4 JDBC Driver 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションのサポートレベルに LocalTransaction または NoTransaction を指定する場合）に選択します。
  HiRDB Type4 JDBC Driver の ConnectionPoolDataSource を使用して HiRDB および XDM/RD E2 に接続します。
- DBConnector_HiRDB_Type4_XA.rar

HiRDB Type4 JDBC Driver 用の DB Connector です。グローバルトランザクションを使用する場合（トランザクションサポートレベルに XATransaction を指定する場合）に選択します。

HiRDB Type4 JDBC Driver の XADataSource を使用して HiRDB に接続します。

- DBConnector_MySQL_CP.rar

  MySQL Connector/J 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションのサポートレベルに LocalTransaction または NoTransaction を指定する場合）に選択します。

  MySQL Connector/J の ConnectionPoolDataSource を使用して MySQL に接続します。

- DBConnector_Oracle_CP.rar

  Oracle JDBC Thin Driver 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションのサポートレベルに LocalTransaction または NoTransaction を指定する場合）に選択します。

  Oracle JDBC Thin Driver の ConnectionPoolDataSource を使用して Oracle に接続します。

- DBConnector_Oracle_XA.rar

  Oracle JDBC Thin Driver 用の DB Connector です。グローバルトランザクションを使用する場合（トランザクションのサポートレベルに XATransaction を指定する場合）に選択します。

  Oracle JDBC Thin Driver の XADataSource を使用して Oracle に接続します。

- DBConnector_PostgreSQL_CP.rar

  PostgreSQL JDBC Driver 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションのサポートレベルに LocalTransaction または NoTransaction を指定する場合）に選択します。

  PostgreSQL JDBC Driver の ConnectionPoolDataSource を使用して PostgreSQL に接続します。

- DBConnector_SQLServer_CP.rar

  SQL Server JDBC Driver 用の DB Connector です。ローカルトランザクションを使用する場合，またはトランザクション管理なしで使用する場合（トランザクションサポートレベルに NoTransaction または LocalTransaction を指定する場合）に選択します。

  SQL Server JDBC Driver の ConnectionPoolDataSource を使用して，SQL Server に接続します。

(2) 実行例

```
cjimportres MyServer -type rar -f DBConnector_HiRDB_Type4_CP.rar
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

4.2.2　DB Connector のプロパティ定義

DB Connector のプロパティを定義します。DB Connector をデプロイしたあとでも実行できます。なお，デプロイ済みの DB Connector のプロパティを変更する場合は，該当する DB Connector を停止した状態で実行してください。

プロパティの設定手順については，「3.5　属性ファイルによるプロパティの設定」を参照してください。次に DB Connector のプロパティ定義について説明します。

(1)　編集する属性ファイル

Connector 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して，DB Connector の Connector 属性ファイルを取得します。

  実行形式

  ```
  cjgetresprop ＜サーバ名称＞ [-nameserver ＜プロバイダURL＞] -type rar -resname ＜DB Connectorの表示名＞ -c ＜Connector属性ファイルパス＞
  ```

  実行例

  ```
  cjgetresprop MyServer -type rar -resname account-rar -c AccountProp.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Connector 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetresprop ＜サーバ名称＞ [-nameserver ＜プロバイダURL＞] -type rar -resname ＜DB Connectorの表示名＞ -c ＜Connector属性ファイルパス＞
  ```

  実行例

  ```
  cjsetresprop MyServer -type rar -resname account-rar -c AccountProp.xml
  ```

  注意事項

  リソースアダプタをデプロイしたあとでプロパティを定義する場合は，cjgetrarprop コマンドと cjsetrarprop コマンドを使用してください。cjgetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetrarprop（RAR ファイルの属性の取得）」を参照してください。cjsetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetrarprop（RAR 属性設定）」を参照してください。

(3)　編集する属性設定項目

DB Connector のプロパティ設定項目を次に示します。

- DB Connector の一般情報
- コンフィグレーションプロパティ
- 実行時プロパティ
- 別名情報

(a)　DB Connector の一般情報

設定できる DB Connector の一般情報属性（<outbound-resourceadapter>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
トランザクションサポートのレベル	○	<transaction-support>
再認証のサポート有無	○	<reauthentication-support>

（凡例）　○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

(b)　コンフィグレーションプロパティ

DB Connector のコンフィグレーションプロパティ（<outbound-resourceadapter> - <connection-definition> - <config-property>タグ）の設定項目を次に示します。

項目	対応するタグ
コンフィグレーションプロパティ名	<config-property-name>
コンフィグレーションプロパティのデータ型	<config-property-type>
コンフィグレーションプロパティの値	<config-property-value>※

注※

コンフィグレーションプロパティの値をクリアする場合，空タグ（<config-property-value></config-property-value>）を指定します。

<config-property-value>タグ自体がない場合は，コンフィグレーションプロパティの値は変更されません。

定義するコンフィグレーションプロパティの数だけ，<config-property>タグ下の設定を繰り返してください。

設定する必要がある項目は，インポートした DB Connector の種類によって一部異なります。<config-property>タグに設定できるプロパティについては，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.10　DB Connector に設定する<config-property>タグに指定できるプロパティ」を参照してください。

なお，<config-property-name>タグに「XAOpenString」および「trustStorePassword」が設定されている場合，cjgetresprop コマンドを実行すると，コンフィグレーションプロパティの値（<config-property-value>）は次のように表示されます。

- プロパティの値が設定されている場合

  コンフィグレーションプロパティの値（<config-property-value>）は取得されないで，「<!-- The config-property-value has already been set. -->」と表示されます。

  「XAOpenString」および「trustStorePassword」の値を変更する場合には，<config-property-value>タグを追加して，変更後の値を設定してください。

- プロパティの値が設定されていない場合

  空タグ（<config-property-value></config-property-value>）が表示されます。

  「XAOpenString」および「trustStorePassword」の値を設定する場合には，<config-property-value>タグに値を追加してください。

ステートメントプーリング機能を使用する場合の，ステートメントプーリングの動作，および注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.4 ステートメントプーリング」を参照してください。

使用する DB Connector のコンフィグレーションプロパティの設定例については，「(4)　コンフィグレーションプロパティの設定例」を参照してください。

(c)　実行時プロパティ

DB Connector の実行時プロパティ（<outbound-resourceadapter> - <connection-definition> - <connector-runtime>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>
プロパティのデータ型	<property-type>
プロパティの値	<property-value>※

注※

プロパティの値をクリアする場合，空タグ（<property-value></property-value>）を指定します。<property-value>タグ自体がない場合は，プロパティの値は変更されません。

定義するプロパティの数だけ，上記の設定を繰り返してください。

プロパティの値（<property-value>）を設定した場合，プロパティ値のデフォルト（<property-default-value>）で設定されている値は無効となります。

プロパティ名（<property-name>）には，次の項目を設定します。

プロパティ項目	プロパティ名（<property-name>）の項目名
ユーザ名※	User
パスワード※	Password
コネクションプールにプールするコネクションの最小値	MinPoolSize
コネクションプールにプールするコネクションの最大値	MaxPoolSize
プール内のコネクションに障害が発生しているかどうかをチェックする方法の選択	ValidationType
プール内のコネクションに障害が発生しているかどうかのチェックを定期的に実行する場合の間隔	ValidationInterval
コネクションの取得に失敗した場合の，再取得する回数	RetryCount
コネクションの取得に失敗した場合の，再取得する間隔	RetryInterval
ログを出力するかどうかの選択	LogEnabled
コネクションの最終利用時刻から，コネクションを自動破棄（コネクションスイーパ）するかを判定するまでの時間	ConnectionTimeout
コネクションの自動破棄（コネクションスイーパ）が動作する間隔	SweeperInterval
コネクション枯渇時にコネクション取得要求をキューで管理するかどうかの選択	RequestQueueEnable
コネクション枯渇時のコネクション取得要求をキューで管理する場合の待ち時間の最大値	RequestQueueTimeout
コネクションプール監視のアラート出力を有効にするかどうかの選択	WatchEnabled
コネクションプールを監視する間隔	WatchInterval
コネクションプール使用状態を監視するしきい値	WatchThreshold
コネクションプール監視結果のファイルを出力するかどうかの選択	WatchWriteFileEnabled
コネクション数調節機能が動作する間隔	ConnectionPoolAdjustmentInterval
コネクションプールのウォーミングアップ機能を有効にするかどうかの選択	Warmup
ネットワーク障害検知機能のタイムアウトを有効にするかどうかの選択	NetworkFailureTimeout

注※

<property-name>タグに「User」,「Password」が設定されている場合，cjgetresprop コマンドを実行すると，プロパティの値（<property-value>）は次のように表示されます。

- プロパティの値が設定されている場合

プロパティの値（<property-value>）は取得されないで，「<!-- The property-value has already been set. -->」と表示されます。

「User」および「Password」の値を変更する場合には，<property-value>タグを追加して，変更後の値を設定してください。

- プロパティの値が設定されていない場合

  空タグ（<property-value></property-value>）が表示されます。

  「User」および「Password」の値を設定する場合には，<property-value>タグに値を追加してください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

コネクションプーリング機能を使用する場合のコネクションプールの動作，および注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.14.1　コネクションプーリング」を参照してください。

(d)　別名情報

DB Connector の別名情報（<outbound-resourceadapter> - <connection-definition> - <connector-runtime> - <resource-external-property>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
リソースの別名	○	<optional-name>
リソース認証方式	△	<res-auth>
リソース共有の有無	△	<res-sharing-scope>

（凡例）　○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

DB Connector の別名の使用法については，「4.7　JNDI 名前空間に登録されるリソースアダプタ名の参照と変更」を参照してください。

(4)　コンフィグレーションプロパティの設定例

次の DB Connector について，コンフィグレーションプロパティの設定例を示します。

- DBConnector_HiRDB_Type4_CP.rar で，HiRDB，または XDM/RD E2 を使用する場合
- DBConnector_HiRDB_Type4_XA.rar で，HiRDB を使用する場合
- DBConnector_MySQL_CP.rar で，MySQL を使用する場合
- DBConnector_Oracle_CP.rar で，Oracle を使用する場合
- DBConnector_Oracle_XA.rar で，Oracle を使用する場合

- DBConnector_PostgreSQL_CP.rar で，PostgreSQL を使用する場合
- DBConnector_SQLServer_CP.rar で，SQL Server を使用する場合

(a) DBConnector_HiRDB_Type4_CP.rar で，HiRDB または XDM/RD E2 を使用する場合

DBConnector_HiRDB_Type4_CP.rar で，HiRDB または XDM/RD E2 を使用する場合の，コンフィグレーションプロパティの設定例を次の表に示します。

表 4-4　データベースとして HiRDB または XDM/RD E2 を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_HiRDB_Type4_CP.rar の場合）

項目名	HiRDB の場合の設定例	XDM/RD E2 の場合の設定例
description	＜HiRDB ポート番号＞※	＜データベースコネクションサーバのサーバスケジュール番号＞※
DBHostName	＜HiRDB ホスト名＞	＜XDM/RD E2 ホスト名＞
environmentVariables	＜HiRDB クライアント環境変数名＞	＜HiRDB クライアント環境変数名＞
loginTimeout	8	8
encodeLang	−	−
JDBC_IF_TRC	false	false
TRC_NO	500	500
uapName	−	−
LONGVARBINARY_Access	REAL	REAL
SQLInNum	300	300
SQLOutNum	300	300
SQLWarningLevel	SQLWARN	SQLWARN
SQLWarningIgnore	false	false
HiRDBCursorMode	false	false
maxBinarySize	0	0
LONGVARBINARY_AccessSize	0	0
LONGVARBINARY_TruncError	true	true
PreparedStatementPoolSize	10	10
CallableStatementPoolSize	10	10
CancelStatement	true	true
logLevel	ERROR	ERROR

（凡例）　－：設定は不要

注※　HiRDB クライアントの環境変数グループ名（Windows の場合）または環境変数グループの設定ファイルのパス（UNIX の場合）も指定できます。

(b)　DBConnector_HiRDB_Type4_XA.rar で，HiRDB を使用する場合

DBConnector_HiRDB_Type4_XA.rar で，HiRDB を使用する場合の，コンフィグレーションプロパティの設定例を次の表に示します。

表 4-5　データベースとして HiRDB を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_HiRDB_Type4_XA.rar の場合）

項目名	HiRDB の場合の設定例
description	＜環境変数グループ識別子＞
DBHostName	＜ HiRDB ホスト名＞
environmentVariables	＜ HiRDB クライアント環境変数名＞
XAOpenString	＜環境変数グループ識別子※1 ＞＋＜環境変数グループの設定ファイルのパス※2 ＞
loginTimeout	8
encodeLang	－
JDBC_IF_TRC	false
TRC_NO	500
uapName	－
LONGVARBINARY_Access	REAL
SQLInNum	300
SQLOutNum	300
SQLWarningLevel	SQLWARN
SQLWarningIgnore	false
HiRDBCursorMode	false
maxBinarySize	0
LONGVARBINARY_AccessSize	0
LONGVARBINARY_TruncError	true
XACloseString	－
XALocalCommitMode	true
PreparedStatementPoolSize	10
CallableStatementPoolSize	10

項目名	HiRDB の場合の設定例
CancelStatement	true
logLevel	ERROR

（凡例）　－：設定は不要

注※1　［Description］フィールドに入力した値を指定します。

注※2　HiRDB の環境変数グループの設定ファイルのパスを指定します。詳細については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.6　データベース接続環境を設定する（HiRDB の設定）」を参照してください。

(c)　DBConnector_MySQL_CP.rar で，MySQL を使用する場合

DBConnector_MySQL_CP.rar で，MySQL を使用する場合のコンフィグレーションプロパティの設定例を次の表に示します。

表 4-6　データベースとして MySQL を使用する場合

項目名	MySQL の場合の設定例
databaseName	＜ MySQL のデータベース名＞
serverName	＜ MySQL のホスト名または IP アドレス＞
portNumber	3306
CancelStatement	true
logLevel	ERROR

(d)　DBConnector_Oracle_CP.rar で，Oracle を使用する場合

DBConnector_Oracle_CP.rar で，Oracle を使用する場合の，コンフィグレーションプロパティの設定例を次の表に示します。

表 4-7　データベースとして Oracle を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_Oracle_CP.rar の場合）

項目名	url で設定しない場合の例	url で設定する場合の例
databaseName	＜ Oracle SID ＞	－
serverName	＜ Oracle のホスト名称，または IP アドレス＞	－
portNumber	1521	－
url	－	jdbc:oracle:thin:@//＜ Oracle のホスト名称，または IP アドレス＞:1521/＜ Oracle SID ＞
loginTimeout	8000	
PreparedStatementPoolSize	10	

項目名	url で設定しない場合の例	url で設定する場合の例
CallableStatementPoolSize	10	
CancelStatement	true	
ConnectionIDUpdate	false	
logLevel	ERROR	

（凡例） －：設定は不要

(e) DBConnector_Oracle_XA.rar で，Oracle を使用する場合

DBConnector_Oracle_XA.rar で，Oracle を使用する場合の，コンフィグレーションプロパティの設定例を次の表に示します。

表 4-8 データベースとして Oracle を使用する場合のコンフィグレーションプロパティの設定例（DBConnector_Oracle_XA.rar の場合）

項目名	url で設定しない場合の例	url で設定する場合の例
databaseName	＜ Oracle SID ＞	－
serverName	＜ Oracle のホスト名称，または IP アドレス＞	－
portNumber	1521	－
url	－	jdbc:oracle:thin:@//＜ Oracle のホスト名称，または IP アドレス＞:1521/＜ Oracle SID ＞
loginTimeout	8000	
sessionTimeout	300	
PreparedStatementPoolSize	10	
CallableStatementPoolSize	10	
CancelStatement	true	
ConnectionIDUpdate	false	
logLevel	ERROR	

（凡例） －：設定は不要

(f) DBConnector_PostgreSQL_CP.rar で，PostgreSQL を使用する場合

DBConnector_PostgreSQL_CP.rar で，PostgreSQL を使用する場合のコンフィグレーションプロパティの設定例を次の表に示します。

表 4-9　データベースとして PostgreSQL を使用する場合

項目名	PostgreSQL の場合の設定例
databaseName	＜ PostgreSQL のデータベース名＞
serverName	＜ PostgreSQL のホスト名または IP アドレス＞
portNumber	5432
PreparedStatementPoolSize	10
CallableStatementPoolSize	10
CancelStatement	true
logLevel	ERROR

(g)　DBConnector_SQLServer_CP.rar で，SQL Server を使用する場合

DBConnector_SQLServer_CP.rar で，SQL Server を使用する場合のコンフィグレーションプロパティの設定例を次の表に示します。

表 4-10　データベースとして SQL Server を使用する場合

項目名	SQL Server の場合の設定例
databaseName	＜ SQL Server のデータベース名＞
serverName	＜ SQL Server のホスト名または IP アドレス＞
applicationName	＜ SQL Server に接続するアプリケーション名＞
instanceName	＜接続する SQL Server のインスタンス名＞
lastUpdateCount	true
lockTimeout	-1
loginTimeout	8
portNumber	1433
selectMethod	cursor
sendStringParametersAsUnicode	true
workstationID	＜アプリケーションサーバのホスト名＞
xopenStates	false
failoverPartner	＜データベースミラーリング構成で使用されるフェイルオーバサーバ名＞
integratedSecurity	false
packetSize	4096
PreparedStatementPoolSize	10
CallableStatementPoolSize	10

項目名	SQL Server の場合の設定例
CancelStatement	true
logLevel	ERROR
applicationIntent	ReadWrite
multiSubnetFailover	false
encrypt	false
hostNameInCertificate	＜SQL Server の TLS/SSL 証明書の検証に使われるホスト名＞
sslProtocol	TLS
trustServerCertificate	false
trustStore	＜trustStore ファイルへのパス名＞
trustStorePassword	＜trustStore データの整合性を確認するために使用するパスワード＞

4.2.3　DB Connector のデプロイ

DB Connector は，デプロイすると J2EE リソースアダプタとして使用できます。**J2EE リソースアダプタ**とは，J2EE サーバに共有スタンドアロンモジュールとして配備したリソースアダプタのことです。サーバ管理コマンドでインポートした，リソースアダプタをデプロイすると，その J2EE サーバ上で動作するすべての J2EE アプリケーションから使用できるようになります。なお，デプロイしたあとで，プロパティを定義することもできます。デプロイ後に定義する場合は，該当する DB Connector を停止した状態で実行してください。プロパティを定義する方法については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

次に示すコマンドを実行して DB Connector をデプロイします。

実行形式

```
cjdeployrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜DB Connector の表示名＞
```

実行例

```
cjdeployrar MyServer -resname account-rar
```

cjdeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeployrar（リソースアダプタのデプロイ）」を参照してください。

4.2.4　DB Connector の接続テスト

DB Connector に設定した情報が正しいかどうか，接続テストによって検証します。

次に示すコマンドを実行して DB Connector の接続テストを実施します。

実行形式

```
cjtestres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -resname <DB Connectorの表示名>
```

実行例

```
cjtestres MyServer -type rar -resname account-rar
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト)」を参照してください。

注意事項

一度接続テストをした DB Connector は，J2EE サーバを再起動するまで削除できません。DB Connector を削除する場合は，DB Connector を停止してから，J2EE サーバを再起動してください。

4.2.5 DB Connector の開始

次に示すコマンドを実行して DB Connector を開始します。

実行形式

```
cjstartrar ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <DB Connectorの表示名>
```

実行例

```
cjstartrar MyServer -resname account-rar
```

cjstartrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartrar（リソースアダプタの開始)」を参照してください。

注意事項

- J2EE アプリケーション中の J2EE リソースが DB Connector を参照している場合は，DB Connector を開始してから，J2EE アプリケーションを開始してください。
- 一度開始した DB Connector は，J2EE サーバを再起動するまで削除できません。DB Connector を削除する場合は，DB Connector を停止してから，J2EE サーバを再起動してください。

4.2.6 DB Connector の停止

次に示すコマンドを実行して DB Connector を停止します。

実行形式

```
cjstoprar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜DB Connectorの表示名＞
```

実行例

```
cjstoprar MyServer -resname account-rar
```

cjstoprar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstoprar（リソースアダプタの停止)」を参照してください。

注意事項

J2EE アプリケーション中の J2EE リソースが DB Connector を参照している場合は，J2EE アプリケーションを停止してから，DB Connector を停止してください。

4.2.7 DB Connector のアンデプロイ

準備

デプロイされている DB Connector を削除する前に，DB Connector を停止して，J2EE サーバを再起動してください。また，DB Connector に対し，一度でも開始または接続テストを試みた場合も同様に J2EE サーバを再起動してください。

次に示すコマンドを実行して DB Connector をアンデプロイします。

実行形式

```
cjundeployrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜DB Connectorの表示名＞
```

実行例

```
cjundeployrar MyServer -resname account-rar
```

cjundeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjundeployrar（リソースアダプタのアンデプロイ)」を参照してください。

4.2.8 DB Connector のエクスポート

DB Connector の内容を RAR ファイルとして出力（エクスポート）します。

次に示すコマンドを実行して DB Connector をエクスポートします。

実行形式

```
cjexportrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -f ＜ファイルパス＞ -res
name ＜DB Connectorの表示名＞
```

実行例

```
cjexportrar MyServer -f res1.rar -resname account-rar
```

cjexportrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjexportrar（リソースアダプタのエクスポート)」を参照してください。

注意事項

Management Server を利用して運用している場合，サーバ管理コマンドと Management Server の実行ホストが異なるときは，DB Connector をエクスポートして，Management Server の実行ホストに格納する必要があります。

4.3　そのほかのリソースと接続するための設定

OpenTP1 などのリソースとの接続には，リソースアダプタを使用します。ここでは，DB Connector，DB Connector for Reliable Messaging，および Reliable Messaging 以外のリソースアダプタと接続するための基本的な設定について説明します。

リソースアダプタは，次の手順で設定します。

1. リソースアダプタをインポートします。
2. プロパティを定義します。
3. リソースアダプタをデプロイします。
   リソースアダプタのデプロイとは，リソースアダプタを J2EE サーバに共有スタンドアロンモジュール（J2EE リソースアダプタ）として配備することです。
4. 接続を確認します。
   正しく接続できるかどうかは，接続テストによって確認できます。

デプロイをした J2EE リソースアダプタは，J2EE アプリケーションのプロパティ設定で，リソースアダプタのリファレンスを解決する必要があります。詳細については，「9.3.3　リソースアダプタのリファレンス定義」を参照してください。

ポイント

そのほかのリソースと接続するリソースアダプタとして，次のリソースアダプタを使用できます。

- Connector 1.0 に準拠するリソースアダプタ
- Connector 1.5 に準拠するリソースアダプタ

Connector 1.0 の仕様に準拠するリソースアダプタのプロパティの定義については，「4.3.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合）」を，Connector 1.5 の仕様に準拠するリソースアダプタのプロパティの定義については，「4.3.3　リソースアダプタのプロパティ定義（Connector 1.5 の場合）」を参照してください。

4.3.1　リソースアダプタのインポート

リソースアダプタをインポートします。

なお，アプリケーションサーバで提供されていない独自のリソースアダプタをインポートする場合は，注意事項に記載されている内容に従ってインポートしてください。

次に示すコマンドを実行してリソースアダプタをインポートします。

実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -f ＜ファイルパス＞
```

実行例

```
cjimportres MyServer -type rar -f mqcadpt.rar
```

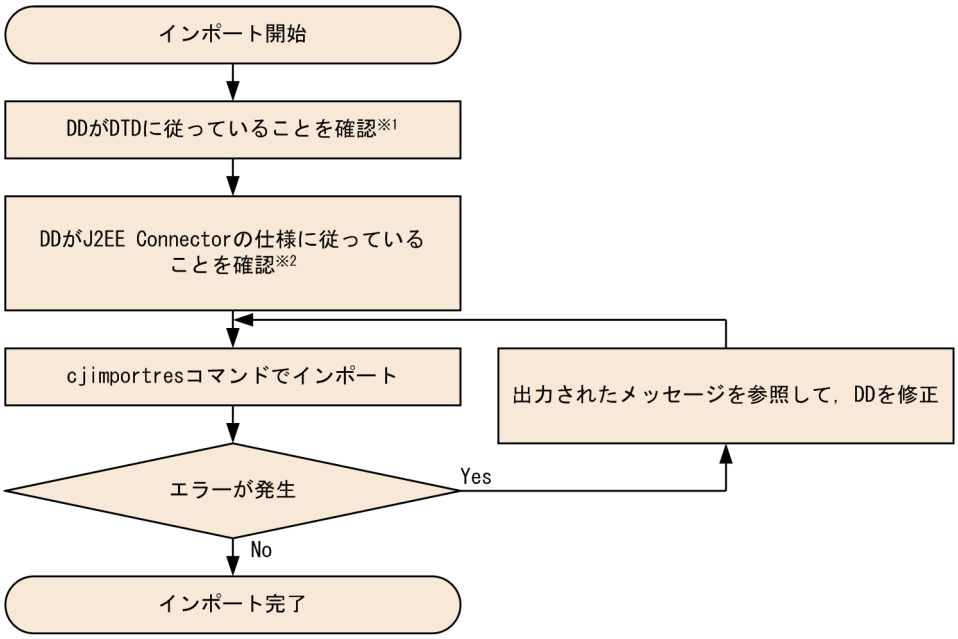
cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

注意事項

- Connector 1.0 以降の仕様に準拠しない DD を持つ RAR はインポートできません。インポートできるのは，Connector 1.0 以降の仕様に準拠した DD を持つ RAR だけです。DD が DTD に従っていることを確認してから，インポートを実行してください。

  インポート手順を次に示します。

  図 4-1　リソースアダプタ（RAR）のインポート手順



注※1

DD は検証済み XML 文書として扱われます。このため，DD 中の DOCTYPE 宣言が誤っていた場合，または DD が DTD の仕様に従っていない場合はエラーになります。

注※2

J2EE Connector の仕様に従った値が DD に指定されていない場合はエラーとなります。詳細は，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「4.1.26　DB Connector を設定する（CUI 利用時)」を参照してください。

また，メッセージが出力された場合は，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照して対処ください。

- インポート時に指定した RAR ファイル名は作業ディレクトリ中のディレクトリ名として使用されます。また，RAR ファイル内の JAR ファイルやネイティブライブラリは作業ディレクトリ内に展開されます。作業ディレクトリのパス長がプラットフォームの上限に達しないように RAR ファイル名および RAR ファイル中のファイル名を指定してください。J2EE アプリケーションを実行するシステムの場合の作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。バッチアプリケーションを実行するシステムの場合の作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.2　バッチサーバの作業ディレクトリ」を参照してください。
- ra.xml の display-name タグが設定されていない場合，RAR ファイル名を基に Display name が付与されます。
- RAR ファイルには，次の名称のファイルを含めないでください。

  rar.properties

  fileinfo.properties

  META-INF/hitachi-ra.xml

4.3.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合）

Connector 1.0 の仕様に準拠するリソースアダプタのプロパティを定義します。リソースアダプタのプロパティ定義は，J2EE リソースアダプタをデプロイしたあとでも実行できます。なお，設定済みのリソースアダプタのプロパティを変更する場合は，該当するリソースアダプタを停止した状態で実行してください。

プロパティの設定手順の概要については，「3.5　属性ファイルによるプロパティの設定」を参照してください。次にリソースアダプタのプロパティ定義について説明します。

(1)　編集する属性ファイル

Connector 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次に示すコマンドを実行して，リソースアダプタの Connector 属性ファイルを取得します。

  実行形式

  ```
  cjgetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -resname
  <リソースアダプタの表示名> -c <Connector属性ファイルパス>
  ```

  実行例

  ```
  cjgetresprop MyServer -type rar -resname Rar1 -c AccountProp.xml
  ```

- **属性の設定**

次に示すコマンドを実行して，Connector 属性ファイルの値を反映します。

実行形式

```
cjsetresprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜リソースアダプタ表示名＞ -c ＜Connector属性ファイルパス＞
```

実行例

```
cjsetresprop MyServer -type rar -resname Rar1 -c AccountProp.xml
```

注意事項

リソースアダプタをデプロイしたあとでプロパティを定義する場合は，cjgetrarprop コマンドと cjsetrarprop コマンドを使用してください。cjgetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetrarprop（RAR ファイルの属性の取得)」を参照してください。cjsetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetrarprop（RAR 属性設定)」を参照してください。

(3)　編集する属性設定項目

リソースアダプタのプロパティ設定項目を次に示します。

- リソースアダプタの一般情報
- コンフィグレーションプロパティ
- 実行時プロパティ
- 別名情報

(a)　リソースアダプタの一般情報

リソースアダプタの一般情報（<outbound-resourceadapter>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
トランザクションサポートのレベル	○	<transaction-support>
再認証のサポート有無	○	<reauthentication-support>

（凡例）　○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

(b)　コンフィグレーションプロパティ

リソースアダプタのコンフィグレーションプロパティ（<outbound-resourceadapter> - <connection-definition> - <config-property>タグ）の設定項目を次に示します。

項目	対応するタグ
コンフィグレーションプロパティ名	<config-property-name>
コンフィグレーションプロパティのデータ型	<config-property-type>
コンフィグレーションプロパティの値	<config-property-value>

定義するコンフィグレーションプロパティの数だけ上記の設定を繰り返してください。

定義するコンフィグレーションプロパティ名（<config-property-name>）については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

(c)　実行時プロパティ

リソースアダプタの実行時プロパティ（<outbound-resourceadapter> - <connection-definition> - <connector-runtime>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>
プロパティのデータ型	<property-type>
プロパティの値	<property-value>

定義するプロパティの数だけ，上記の設定を繰り返してください。

定義する実行時プロパティ名（<property-name>）およびコネクションプールの動作と注意事項については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

(d)　別名情報

リソースアダプタの別名情報（<outbound-resourceadapter> - <connection-definition> - <connector-runtime> - <resource-external-property>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
リソースの別名	○	<optional-name>
リソース認証方式	△	<res-auth>
リソース共有の有無	△	<res-sharing-scope>

（凡例）　○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

リソースアダプタの別名の使用法については，「4.7　JNDI 名前空間に登録されるリソースアダプタ名の参照と変更」を参照してください。

4.3.3　リソースアダプタのプロパティ定義（Connector 1.5 の場合）

Connector 1.5 の仕様に準拠するリソースアダプタに接続する場合のプロパティを定義します。

リソースアダプタのプロパティ定義は，J2EE リソースアダプタをデプロイしたあとでも実行できます。なお，リソースアダプタの設定済みのプロパティを変更する場合は，該当するリソースアダプタを停止した状態で実行してください。

プロパティの設定手順の概要については，「3.5　属性ファイルによるプロパティの設定」を参照してください。次にリソースアダプタのプロパティ定義について説明します。

(1)　編集する属性ファイル

Connector 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して，リソースアダプタの Connector 属性ファイルを取得します。

  実行形式

  ```
  cjgetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -resname
  <リソースアダプタの表示名> -c <Connector属性ファイルパス>
  ```

  実行例

  ```
  cjgetresprop MyServer -type rar -resname Rar1 -c AccountProp.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Connector 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -resname
  <リソースアダプタ表示名> -c <Connector属性ファイルパス>
  ```

  実行例

  ```
  cjsetresprop MyServer -type rar -resname Rar1 -c AccountProp.xml
  ```

  注意事項

  リソースアダプタをデプロイしたあとでプロパティを定義する場合は，cjgetrarprop コマンドと cjsetrarprop コマンドを使用してください。cjgetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetrarprop（RAR ファイルの属性の取得）」を参照してください。cjsetrarprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetrarprop（RAR 属性設定）」を参照してください。

(3)　編集する属性設定項目

Connector 1.5 の仕様に準拠するリソースアダプタは，コネクション定義を複数持つ場合があります。

プロパティ項目は，次のように分けて設定します。

- **リソースアダプタのプロパティ設定**
  次のプロパティを設定します。
  - コンフィグレーションプロパティ
  - Outbound リソースアダプタのプロパティ
  - Inbound リソースアダプタのプロパティ
  - 管理対象オブジェクトのプロパティ
  - 実行時プロパティ

  プロパティ設定項目については，「(4)　リソースアダプタのプロパティ設定」を参照してください。
- **Outbound リソースアダプタのコネクション単位のプロパティ設定**
  Outbound リソースアダプタを使用する場合は，次のプロパティを設定します。
  - コンフィグレーションプロパティ
  - 実行時プロパティ
  - 別名情報

  Outbound リソースアダプタのコネクション単位のプロパティ設定項目については，「(5)　Outbound リソースアダプタのコネクション単位のプロパティ設定」を参照してください。

(4)　リソースアダプタのプロパティ設定

リソースアダプタのプロパティ設定項目を次に示します。

(a)　コンフィグレーションプロパティ

リソースアダプタのコンフィグレーションプロパティ（<resourceadapter> - <config-property>タグ）の設定項目を次に示します。

項目	対応するタグ
コンフィグレーションプロパティ名	<config-property-name>
コンフィグレーションプロパティのデータ型	<config-property-type>
コンフィグレーションプロパティの値	<config-property-value>

定義するコンフィグレーションプロパティの数だけ上記の設定を繰り返してください。

定義するコンフィグレーションプロパティ名（<config-property-name>）については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

(b) Outbound リソースアダプタのプロパティ

Outbound リソースアダプタのプロパティ（<outbound-resourceadapter>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
コネクション単位でのプロパティ設定※	△	<connection-definition>
トランザクションサポートのレベル	○	<transaction-support>
再認証のサポート有無	○	<reauthentication-support>

（凡例） ○：必須　△：任意

注※　Outbound リソースアダプタのコネクション単位のプロパティ設定については，「(5)　Outbound リソースアダプタのコネクション単位のプロパティ設定」を参照してください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

(c) Inbound リソースアダプタのプロパティ

Inbound リソースアダプタのプロパティ（<inbound-resourceadapter> - <messageadapter> - <messagelistener>タグ）の設定項目を次に示します。

項目	対応するタグ
メッセージリスナのタイプ	<messagelistener-type>
ActivationSpec の情報	<activationspec>

(d) 管理対象オブジェクトのプロパティ

リソースアダプタの管理対象オブジェクトのプロパティ（<adminobject> - <config-property>タグ）の設定項目を次に示します。

項目	対応するタグ
管理対象オブジェクトのプロパティ名	<config-property-name>
管理対象オブジェクトのプロパティのデータ型	<config-property-type>
管理対象オブジェクトのプロパティの値	<config-property-value>

(e) 実行時プロパティ

リソースアダプタの実行時プロパティ（<resourceadapter-runtime> - <property>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>

項目	対応するタグ
プロパティのデータ型	<property-type>
プロパティの値	<property-value>※

注※　プロパティの値の設定方法については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

定義するプロパティの数だけ，上記の設定を繰り返してください。

プロパティ名（<property-name>）には，次の項目を設定します。

プロパティ項目	プロパティ名（<property-name>）の項目名
スレッドプールで同時に実行される最大スレッド数	MaxTPoolSize
スレッドプールに存在する最小スレッド数	MinTPoolSize
スレッドプールのスレッド解放までのタイムアウト値(秒)	TPoolKeepalive

注　ライフサイクル管理機能が有効な場合に設定します。ライフサイクル管理機能を使用していない（<resourceadapter-class>を指定していない）場合，プロパティ値（<property-value>）は無視されます。ライフサイクル管理機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.16.1　リソースアダプタのライフサイクル管理」を参照してください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

(5)　Outbound リソースアダプタのコネクション単位のプロパティ設定

Outbound リソースアダプタのコネクション単位のプロパティ設定項目を次に示します。

(a)　コンフィグレーションプロパティ

Outbound リソースアダプタのコンフィグレーションプロパティ（<outbound-resourceadapter> - <connection-definition> - <config-property>タグ）の設定項目を次に示します。

項目	対応するタグ
コンフィグレーションプロパティ名	<config-property-name>
コンフィグレーションプロパティのデータ型	<config-property-type>
コンフィグレーションプロパティの値	<config-property-value>

定義するコンフィグレーションプロパティの数だけ上記の設定を繰り返してください。

定義するコンフィグレーションプロパティ名（<config-property-name>）については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

なお，コネクション単位のコンフィグレーションプロパティには，コネクション単位に設定されたコンフィグレーションプロパティ（<outbound-resourceadapter> - <connection-definition> - <config-

property>タグ）とリソースアダプタに設定されたコンフィグレーションプロパティ（<resourceadapter> - <config-property>タグ）を合わせたものが設定されます。同じコンフィグレーションプロパティ名が設定されている場合は，コネクション単位に設定されたコンフィグレーションプロパティが優先されます。

(b)　実行時プロパティ

Outbound リソースアダプタの実行時プロパティ（<outbound-resourceadapter> - <connection-definition> - <connector-runtime>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>
プロパティのデータ型	<property-type>
プロパティの値	<property-value>

定義するプロパティの数だけ，上記の設定を繰り返してください。

定義する実行時プロパティ名（<property-name>）およびコネクションプールの動作と注意事項については，「4.2.2　DB Connector のプロパティ定義」を参照してください。

(c)　別名情報

Outbound リソースアダプタの別名情報（<outbound-resourceadapter> - <connection-definition> - <connector-runtime> - <resource-external-property>タグ）の設定項目を次に示します。

項目	必須	対応するタグ
リソースの別名	○	<optional-name>
リソース認証方式	△	<res-auth>
リソース共有の有無	△	<res-sharing-scope>

（凡例）　○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.1　Connector 属性ファイルの指定内容」を参照してください。

リソースアダプタの別名の使用方法については，「4.7　JNDI 名前空間に登録されるリソースアダプタ名の参照と変更」を参照してください。

4.3.4　リソースアダプタのデプロイ

リソースアダプタは，デプロイすると J2EE リソースアダプタとして使用できます。

サーバ管理コマンドでインポートしたリソースアダプタをデプロイすると，その J2EE サーバ上で動作するすべての J2EE アプリケーションから使用できるようになります。なお，リソースアダプタをデプロイ

したあとで，リソースアダプタのプロパティを定義することもできます。プロパティを定義する方法については，リソースアダプタが対応しているコネクタアーキテクチャの仕様に応じて，「4.3.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合）」または「4.3.3　リソースアダプタのプロパティ定義（Connector 1.5 の場合）」を参照してください。

次に示すコマンドを実行してリソースアダプタをデプロイします。

実行形式

```
cjdeployrar ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <リソースアダプタの表示名> ［-resname <リソースアダプタの表示名>］
```

実行例

```
cjdeployrar MyServer -resname Rar1
```

cjdeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeployrar（リソースアダプタのデプロイ）」を参照してください。

注意事項

開始状態の実行時情報を含んだリソースアダプタを指定すると，デプロイ完了後に自動開始されます。自動開始処理に失敗した場合でも，この J2EE リソースアダプタは削除されません。

4.3.5　J2EE リソースアダプタの接続テスト

リソースアダプタに設定した情報が正しいかどうか，接続テストによって検証します。

次に示すコマンドを実行してリソースアダプタの接続テストを実施します。

実行形式

```
cjtestres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type rar -resname <リソースアダプタの表示名> ［-resname <リソースアダプタの表示名>］
```

実行例

```
cjtestres MyServer -type rar -resname Rar1
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト）」を参照してください。

注意事項

- 一度接続テストをしたリソースアダプタは，J2EE サーバを再起動するまで削除できません。リソースアダプタを削除する場合は，リソースアダプタを停止してから，J2EE サーバを再起動してください。
- Connector 1.5 の仕様に準拠するリソースアダプタにコネクション定義が複数ある場合，すべてのコネクション定義に対して接続テストが実行されます。コネクション定義のどれかでエラーが発生

した場合でも，すべてのコネクション定義に対して接続テストが実行されます。ただし，その場合，接続テスト結果は「失敗」となります。

- Connector 1.5 の仕様に準拠するリソースアダプタの場合は，Outbound リソースアダプタを含むときだけ，接続テストを実施できます。

4.3.6 J2EE リソースアダプタの開始

J2EE リソースアダプタを開始します。

次に示すコマンドを実行して J2EE リソースアダプタを開始します。

実行形式

```
cjstartrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjstartrar MyServer -resname Rar1
```

cjstartrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartrar（リソースアダプタの開始)」を参照してください。

注意事項

- J2EE アプリケーション中の J2EE リソースが J2EE リソースアダプタを参照している場合は，J2EE リソースアダプタを開始してから，J2EE アプリケーションを開始してください。
- 一度開始した J2EE リソースアダプタは，J2EE サーバを再起動するまで削除できません。J2EE リソースアダプタを削除するには，J2EE リソースアダプタを停止してから，J2EE サーバを再起動し，削除しようとする J2EE リソースアダプタが J2EE アプリケーションの参照解決の対象外であることを確認してください。

4.3.7 J2EE リソースアダプタの停止

J2EE リソースアダプタを停止します。

次に示すコマンドを実行して J2EE リソースアダプタを停止します。

実行形式

```
cjstoprar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjstoprar MyServer -resname Rar1
```

cjstoprar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstoprar（リソースアダプタの停止)」を参照してください。

注意事項

J2EE アプリケーション中の J2EE リソースが J2EE リソースアダプタを参照している場合は，J2EE アプリケーションを停止してから，J2EE リソースアダプタを停止してください。

4.3.8 J2EE リソースアダプタのアンデプロイ

デプロイされている J2EE リソースアダプタを削除します。

準備

J2EE リソースアダプタを削除する前に，J2EE リソースアダプタを停止して J2EE サーバを再起動してください。また，J2EE リソースアダプタに対し，一度でも開始または接続テストを試みた場合も同様に J2EE サーバを再起動してください。

次に示すコマンドを実行して J2EE リソースアダプタをアンデプロイします。

実行形式

```
cjundeployrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjundeployrar MyServer -resname Rar1
```

cjundeployrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjundeployrar（リソースアダプタのアンデプロイ)」を参照してください。

4.3.9 J2EE リソースアダプタのエクスポート

J2EE リソースアダプタの内容を RAR ファイルとして出力（エクスポート）します。

次に示すコマンドを実行して J2EE リソースアダプタをエクスポートします。

実行形式

```
cjexportrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -f ＜ファイルパス＞ -resname ＜J2EEリソースアダプタの表示名＞
```

実行例

```
cjexportrar MyServer -f res1.rar -resname Rar1
```

cjexportrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjexportrar（リソースアダプタのエクスポート）」を参照してください。

注意事項

Management Server を利用して運用している場合，サーバ管理コマンドと Management Server の実行ホストが異なる場合，J2EE リソースアダプタをエクスポートして，Management Server の実行ホストに格納する必要があります。

4.4　J2EE リソースアダプタの状態と一覧の参照

デプロイされているリソースアダプタのアダプタ名と状態を参照できます。また，Connector 1.5 の仕様に準拠するリソースアダプタの場合，コネクション定義識別子の一覧や，メッセージリスナのタイプの一覧などの情報も参照できます。

4.4.1　J2EE リソースアダプタの状態の参照

次に示すコマンドを実行して，デプロイされているすべてのリソースアダプタの名称と状態（開始状態または停止状態）を参照します。

実行形式

```
cjlistrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ ［-spec］
```

実行例

```
cjlistrar MyServer
```

-spec オプションを指定すると，リソースアダプタの状態表示一覧にコネクタアーキテクチャの仕様バージョンが表示されます。

cjlistrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示)」を参照してください。

4.4.2　コネクション定義識別子の一覧の参照（Outbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，デプロイされている Outbound リソースアダプタのコネクション定義識別子の一覧を参照できます。

実行形式

```
cjlistrar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜RARの表示名＞ -outbound
```

実行例

```
cjlistrar MyServer -resname account-rar -outbound
```

cjlistrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示)」を参照してください。

4.4.3 メッセージリスナのタイプの一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，デプロイされている Inbound リソースアダプタのメッセージリスナのタイプの一覧を参照できます。

実行形式

```
cjlistrar ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <RARの表示名> -inbound
```

実行例

```
cjlistrar MyServer -resname account-rar -inbound
```

cjlistrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示）」を参照してください。

4.4.4 メッセージリスナのアクティブ化に必要なプロパティ名の一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，デプロイされている Inbound リソースアダプタのメッセージリスナのアクティブ化に必要なプロパティ名の一覧を参照できます。

実行形式

```
cjlistrar ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <RARの表示名> -listenertype <メッセージリスナのタイプ名>
```

実行例

```
cjlistrar MyServer -resname account-rar -listenertype javax.jms.MessageListener
```

cjlistrar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistrar（リソースアダプタの一覧表示）」を参照してください。

4.5　リソースアダプタの一覧の参照

インポートされているリソースアダプタの一覧を参照できます。また，Connector 1.5 の仕様に準拠するリソースアダプタの場合，コネクション定義識別子の一覧や，メッセージリスナのタイプの一覧などの情報も参照できます。

4.5.1　リソースアダプタの一覧の参照

次に示すコマンドを実行して，インポートされているリソースアダプタの一覧を参照します。

実行形式

```
cjlistres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar ［-spec］
```

実行例

```
cjlistres MyServer -type rar
```

-spec オプションを指定すると，一覧にコネクタアーキテクチャの仕様バージョンが表示されます。

cjlistres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistres（リソースの一覧表示)」を参照してください。

4.5.2　コネクション定義識別子の一覧の参照（Outbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE リソースアダプタとしてデプロイする Outbound リソースアダプタのコネクション定義識別子の一覧を参照します。

実行形式

```
cjlistres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜RARの表示名＞ -outbound
```

実行例

```
cjlistres MyServer -type rar -resname account-rar -outbound
```

cjlistres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistres（リソースの一覧表示)」を参照してください。

4.5.3　メッセージリスナのタイプの一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE リソースアダプタとしてデプロイする Inbound リソースアダプタのメッセージリスナのタイプの一覧を参照できます。

実行形式

```
cjlistres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜RARの表示名＞ -inbound
```

実行例

```
cjlistres MyServer -type rar -resname account-rar -inbound
```

cjlistres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistres（リソースの一覧表示)」を参照してください。

4.5.4　メッセージリスナのアクティブ化に必要なプロパティ名の一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE リソースアダプタとしてデプロイする Inbound リソースアダプタのメッセージリスナのアクティブ化に必要なプロパティ名の一覧を参照できます。

実行形式

```
cjlistres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜RARの表示名＞ -listenertype ＜メッセージリスナのタイプ名＞
```

実行例

```
cjlistres MyServer -type rar -resname account-rar -listenertype javax.jms.MessageListener
```

cjlistres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistres（リソースの一覧表示)」を参照してください。

4.6　コネクションプールの状態の確認

リソースアダプタのコネクションプールの情報およびコネクションの状態（使用中，未使用）を参照します。また，必要に応じて，リソースアダプタのコネクションを削除します。

4.6.1　コネクションプールの状態表示

次に示すコマンドを使用して，リソースアダプタのコネクションプールの情報および状態を表示します。

実行形式

```
cjlistpool ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjlistpool MyServer -resname Rar1
```

cjlistpool コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistpool（コネクションプールの一覧表示)」を参照してください。

参考

Connector 1.5 の仕様に準拠するリソースアダプタの場合，-resname オプションの＜リソースアダプタの表示名＞は次の形式で指定します。

＜リソースアダプタの表示名>!<コネクション定義識別子>

4.6.2　コネクションプールの削除

次に示すコマンドを使用して，リソースアダプタのコネクションプールを削除します。未使用のコネクションプールはすべて削除されます。

実行形式

```
cjclearpool ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type connector -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjclearpool MyServer -type connector -resname Rar1
```

cjclearpool コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclearpool（コネクションプール内のコネクション削除)」を参照してください。

参考

Connector 1.5 の仕様に準拠するリソースアダプタの場合，-resname オプションの＜リソースアダプタの表示名＞は次の形式で指定します。

＜リソースアダプタの表示名>!<コネクション定義識別子>

4.7　JNDI 名前空間に登録されるリソースアダプタ名の参照と変更

JNDI 名前空間に登録されるリソースアダプタ名を変更します。

リソースアダプタ名には別名も付けられます。別名を付けることで，JNDI 名前空間から任意の名前でリソースアダプタを参照できるようになります。なお，この機能を**ユーザ指定名前空間機能**といいます。

リソースアダプタの JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。ユーザ指定名前空間機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.6　Enterprise Bean または J2EE リソースへの別名付与（ユーザ指定名前空間機能)」を参照してください。

別名は，Connector 属性ファイルの<resource-external-property>タグで追加できます。ただし，JMS インタフェースを使用するリソースアダプタには，別名は付与できません。

Connector 属性ファイルの<resource-external-property>タグの編集については，使用するリソースアダプタの種類に応じて，「4.2.2　DB Connector のプロパティ定義」，「4.3.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合)」または「4.3.3　リソースアダプタのプロパティ定義（Connector 1.5 の場合)」の別名情報の設定を参照してください。

注意事項

開始状態の J2EE アプリケーションがある場合，別名を設定しているリソースアダプタの停止および削除はできません。J2EE サーバで開始されているすべての J2EE アプリケーションを停止させてください。

4.8　リソースアダプタの削除

次に示すコマンドを実行してリソースアダプタを削除します。

実行形式

```
cjdeleteres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjdeleteres MyServer -type rar -resname account-rar
```

cjdeleteres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeleteres（リソースの削除)」を参照してください。

注意事項

cosminexus.xml を含むアプリケーションから J2EE リソースを削除した場合，cosminexus.xml に含まれている削除対象の J2EE リソースに関するアプリケーションサーバ独自の定義情報は削除されません。cosminexus.xml を含むアプリケーションから J2EE リソースを削除する場合の詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16.3.6　cosminexus.xml を含むアプリケーションの運用」を参照してください。

4.9 リソースアダプタのコピー

次に示すコマンドを実行してリソースアダプタのプロパティをコピーします。

実行形式

```
cjcopyres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -src ＜コピー元のリソースアダプタの表示名＞ -dst ＜コピー先のリソースアダプタの表示名＞
```

実行例

```
cjcopyres MyServer -type rar -src account-rar -dst account-rar2
```

cjcopyres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjcopyres（リソースのコピー)」を参照してください。

5 J2EE アプリケーションに含まれるリソースアダプタの設定

この章では，J2EE アプリケーションに含まれるリソースアダプタのアプリケーション設定操作について説明します。

5.1　J2EE アプリケーションに含まれるリソースアダプタの設定の概要

J2EE アプリケーションに含まれるリソースアダプタの設定とは，J2EE アプリケーションに含まれるリソースアダプタを，その J2EE アプリケーションで利用できる状態にするための操作です。

J2EE アプリケーションに含めて使用できるリソースアダプタの種類と，アプリケーション設定操作の概要について説明します。

5.1.1　利用できるリソースアダプタ

J2EE アプリケーションに含めて利用できるリソースアダプタを次に示します。

- DB Connector
- TP1 Connector
- そのほかの Connector 1.0 に準拠したリソースアダプタ，または Connector 1.5 に準拠したリソースアダプタ※

注※　そのほかの Connector 1.0 に準拠したリソースアダプタ，または Connector 1.5 に準拠したリソースアダプタについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「3.3.2　リソースアダプタの種類」を参照してください。

注意事項

J2EE リソースアダプタと J2EE アプリケーションに含まれるリソースアダプタは，J2EE アプリケーションから同時に使用できます。ただし，J2EE リソースアダプタと J2EE アプリケーションに含まれるリソースアダプタを，同じ表示名で同時に使用することはできません。同じ表示名に設定するとエラーとなります。

5.1.2　設定する項目と操作の概要

J2EE アプリケーションに含まれるリソースアダプタの設定の概要について，次の表に示します。

表 5-1　J2EE アプリケーションに含まれるリソースアダプタの設定の概要

設定項目		内容	参照先
リソースアダプタを含んだ J2EE アプリケーションの作成※	J2EE アプリケーションへのリソースアダプタの追加	RAR ファイルを J2EE サーバにインポートして，J2EE アプリケーションに追加します。	5.2
	リソースアダプタを含む J2EE アプリケーションのインポート	アプリケーション開発環境で作成した，RAR ファイルを含んだ J2EE アプリケーションを J2EE サーバにインポートします。	5.3

設定項目	内容	参照先
リソースアダプタのプロパティ定義	データベースとの接続についての情報を設定します。 • リソースアダプタの一般情報 • コンフィグレーションプロパティ • 実行時プロパティ • 別名情報	5.4
リソースアダプタの接続テスト	リソースアダプタに設定した内容が正しいかどうかを検証します。	5.5
リソースアダプタを含むJ2EEアプリケーションの開始と停止	リソースアダプタを含むJ2EEアプリケーションを開始または停止します。	5.6
J2EEアプリケーションに含まれるリソースアダプタの一覧の参照	J2EEアプリケーションに含まれるリソースアダプタの一覧，コネクション定義識別子の一覧などを参照します。	5.7
コネクションプールの一覧表示と削除	J2EEアプリケーションに含まれるリソースアダプタのコネクションプールの状態を参照します。また，必要に応じてコネクションを削除します。	5.8

注※　リソースアダプタを含んだJ2EEアプリケーションは次のどちらかの方法で作成します。

- J2EEアプリケーションにリソースアダプタを追加します。
  EJB-JARまたはWARをJ2EEアプリケーションに追加する場合と同じ方法で，RARをJ2EEアプリケーションに追加します。
- リソースアダプタを含むJ2EEアプリケーションをインポートします。
  アプリケーション開発環境でリソースアダプタを含むJ2EEアプリケーションを成して，そのJ2EEアプリケーションをJ2EEサーバにインポートします。

5.2　J2EE アプリケーションへのリソースアダプタの追加

J2EE アプリケーションにリソースアダプタを追加します。

追加を実行する前に，追加するリソースアダプタと同じ表示名，または同じ別名のリソースアダプタが，デプロイされていないことを確認してください。同じ表示名，または同じ別名のリソースアダプタがデプロイされている場合，そのリソースアダプタは J2EE アプリケーションに追加できません。なお，別の J2EE アプリケーションに含まれるリソースアダプタと同じ表示名，別名は使用できます。

手順を次に示します。

1. リソースアダプタの RAR ファイルを J2EE サーバにインポートします。

   J2EE アプリケーションに追加する RAR ファイルのインポートについては，「7.2.3　リソースアダプタ（RAR）のインポート」を参照してください。

2. J2EE アプリケーションにインポートした RAR を追加します。

   J2EE アプリケーションへのリソースアダプタ（RAR）の追加については，「7.2.4　J2EE アプリケーションの新規作成」の「(3)　RAR ファイルの追加」を参照してください。

5.3　リソースアダプタを含む J2EE アプリケーションのインポート

リソースアダプタを含む J2EE アプリケーションをインポートします。

インポートを実行する前に，J2EE アプリケーションに含まれるリソースアダプタと同じ表示名，または同じ別名のリソースアダプタが，デプロイされていないことを確認してください。同じ表示名，または同じ別名のリソースアダプタがデプロイされている場合，そのリソースアダプタを含む J2EE アプリケーションはインポートできません。なお，別の J2EE アプリケーションに含まれるリソースアダプタと同じ表示名，別名は使用できます。

次の J2EE アプリケーションをインポートします。

- アーカイブ形式の J2EE アプリケーション

  J2EE サーバの作業ディレクトリ以下に，EJB-JAR/WAR/RAR ファイルのコンポーネントを持つ J2EE アプリケーションです。アーカイブ形式の J2EE アプリケーションのインポートについては，「8.1.1 アーカイブ形式の J2EE アプリケーションのインポート」を参照してください。
- 展開ディレクトリ形式の J2EE アプリケーション

  EJB-JAR や WAR などのアプリケーションの実体を，J2EE サーバの外部にある一定のルールに従ったファイル／ディレクトリに持つ J2EE アプリケーションです。ただし，展開ディレクトリ形式の J2EE アプリケーションに含まれる RAR ファイルは展開ファイル形式ではなく，アーカイブ形式で格納されます。展開ディレクトリ形式の J2EE アプリケーションのインポートについては，「8.1.2　展開ディレクトリ形式の J2EE アプリケーションのインポート」を参照してください。

5.4　リソースアダプタのプロパティ定義

J2EE アプリケーションに含まれるリソースアダプタのプロパティを設定します。

リソースアダプタのプロパティの設定は，リソースアダプタを含む J2EE アプリケーションを停止した状態で実行してください。

プロパティの設定手順については，「3.5　属性ファイルによるプロパティの設定」を参照してください。

5.4.1　編集する属性ファイル

次のどちらかの属性ファイルを利用して，J2EE アプリケーションに含まれるリソースアダプタのプロパティを設定できます。

- Connector 属性ファイルを利用します。
- アプリケーション統合属性ファイルを利用します。

  J2EE アプリケーション統合属性ファイルを構成するコンポーネントの属性ファイルのうち，Connector 属性の要素でプロパティを設定します。

J2EE アプリケーションを構成するコンポーネントのプロパティの設定では，Connector 属性ファイルとアプリケーション統合属性ファイルを個々に利用することも，あわせて利用することもできます。

この章では，Connector 属性ファイルを使用してのプロパティ設定について説明しています。アプリケーション統合属性ファイルを使用してのプロパティ設定の手順については，「9.2　アプリケーション統合属性ファイルによるプロパティ設定」を参照してください。

5.4.2　編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次に示すコマンドを実行してリソースアダプタの属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type rar -resname <リソースアダプタ表示名> -c <Connctor属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name App1 -type rar -resname account-rar -c AccountProp.xml
  ```

- **属性の設定**

  次に示すコマンドを実行して，リソースアダプタの属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type rar -resname ＜リソースアダプタ表示名＞ -c ＜Connector属性ファイルパス＞
```

実行例

```
cjsetappprop MyServer -name App1 -type rar -resname account-rar -c AccountProp.xml
```

5.4.3　編集する属性設定項目

リソースアダプタの種類に応じて，プロパティを定義します。

- Connector 1.0 の仕様に準拠するリソースアダプタの場合

  Connector 1.0 の仕様に準拠するリソースアダプタの属性設定項目については，「4.3.2　リソースアダプタのプロパティ定義（Connector 1.0 の場合）」の「(3)　編集する属性設定項目」を参照してください。

- Connector 1.5 の仕様に準拠するリソースアダプタの場合

  Connector 1.5 の仕様に準拠するリソースアダプタの属性設定項目については，「4.3.3　リソースアダプタのプロパティ定義（Connector 1.5 の場合）」の「(3)　編集する属性設定項目」を参照してください。

5.5　リソースアダプタの接続テスト

J2EE アプリケーションに含まれるリソースアダプタに設定した情報が正しいかどうか，接続テストによって検証します。

次に示すコマンドを実行して J2EE アプリケーションに含まれるリソースアダプタの接続テストを実行します。

実行形式

```
cjtestres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type rar -resname ＜リソースアダプタの表示名＞
```

実行例

```
cjtestres -name App1 -type rar -resname DB_Connector_for_Cosminexus_Driver
```

J2EE アプリケーションに含まれるリソースアダプタを接続テストする場合の注意事項については，次の説明を参照してください。

- データベースと接続するための設定（DB Connector を使用する場合）
  「4.2.4　DB Connector の接続テスト」
- そのほかのリソースと接続するための設定（ほかのリソースアダプタを使用する場合）
  「4.3.5　J2EE リソースアダプタの接続テスト」

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト）」を参照してください。

5.6　リソースアダプタを含む J2EE アプリケーションの開始と停止

J2EE アプリケーションを開始すると，J2EE アプリケーションに含まれるすべてのリソースアダプタは開始されます。また，J2EE アプリケーションを停止すると，J2EE アプリケーションに含まれるすべてのリソースアダプタは停止されます。リソースアダプタの開始順序および停止順序の制御はできません。

J2EE アプリケーションの開始については，「10.2.1　J2EE アプリケーションの開始」を参照してください。

J2EE アプリケーションの停止については，「10.2.2　J2EE アプリケーションの停止」を参照してください。

注意事項

J2EE アプリケーションがリソースアダプタを含み，application.xml の<module>タグ以下で RAR を記述している場合，EJB-JAR や WAR が，リソースアダプタを参照していなくても，J2EE アプリケーション開始時にリソースアダプタは開始されます。

5.7　J2EE アプリケーションに含まれるリソースアダプタの一覧の参照

J2EE アプリケーションに含まれるリソースアダプタの一覧を参照できます。また，Connector 1.5 の仕様に準拠するリソースアダプタの場合，コネクション定義識別子の一覧や，メッセージリスナのタイプの一覧などの情報も参照できます。

5.7.1　リソースアダプタの一覧の参照

次に示すコマンドを実行して，J2EE アプリケーションに含まれるリソースアダプタの一覧を参照します。

実行形式

```
cjlistapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name
＜J2EEアプリケーション名＞ -type rar ［-spec］
```

実行例

```
cjlistapp MyServer -name App1 -type rar
```

-spec オプションを指定すると，一覧にコネクタアーキテクチャの仕様バージョンが表示されます。

cjlistapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistapp（アプリケーションの一覧表示)」を参照してください。

5.7.2　コネクション定義識別子の一覧の参照（Outbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE アプリケーションに含まれる Outbound リソースアダプタのコネクション定義識別子の一覧を参照します。

実行形式

```
cjlistapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type rar -resname ＜リソースアダプタの表示名＞ -outbound
```

実行例

```
cjlistapp MyServer -name App1 -type rar -resname account-rar -outbound
```

cjlistapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistapp（アプリケーションの一覧表示)」を参照してください。

5.7.3　メッセージリスナのタイプの一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE アプリケーションに含まれる Inbound リソースアダプタのメッセージリスナのタイプの一覧を参照します。

実行形式

```
cjlistapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type rar -resname ＜リソースアダプタの表示名＞ -inbound
```

実行例

```
cjlistapp MyServer -name App1 -type rar -resname account-rar -inbound
```

cjlistapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistapp（アプリケーションの一覧表示)」を参照してください。

5.7.4　メッセージリスナのアクティブ化に必要なプロパティ名の一覧の参照（Inbound リソースアダプタ）

Connector 1.5 の仕様に準拠するリソースアダプタの場合，次に示すコマンドを実行して，J2EE アプリケーションに含まれる Inbound リソースアダプタのメッセージリスナのアクティブ化に必要なプロパティ名の一覧を参照します。

実行形式

```
cjlistapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type rar -resname ＜リソースアダプタの表示名＞ -listenertype ＜メッセージリスナのタイプ名＞
```

実行例

```
cjlistapp MyServer -name App1 -type rar -resname account-rar -listenertype javax.jms.MessageListener
```

cjlistapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistapp（アプリケーションの一覧表示)」を参照してください。

5.8　コネクションプールの一覧表示と削除

J2EE アプリケーションに含まれるリソースアダプタの，コネクションプールの情報およびコネクションの状態（使用中，未使用）を参照します。また，必要に応じて，リソースアダプタのコネクションを削除します。

5.8.1　コネクションプールの状態表示

次に示すコマンドを使用して，J2EE アプリケーションに含まれるリソースアダプタの，コネクションプールの情報および状態を表示します。

実行形式

```
cjlistpool ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -resname <リソースアダプタの表示名>
```

実行例

```
cjlistpool MyServer -name App1 -resname Rar1
```

cjlistpool コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistpool（コネクションプールの一覧表示)」を参照してください。

参考

Connector 1.5 の仕様に準拠するリソースアダプタの場合，-resname オプションの<リソースアダプタの表示名>は次の形式で指定します。

<リソースアダプタの表示名>!<コネクション定義識別子>

5.8.2　コネクションプールの削除

次に示すコマンドを使用して，必要に応じて，J2EE アプリケーションに含まれるリソースアダプタのコネクションプールを削除します。なお，未使用のコネクションプールはすべて削除されます。

実行形式

```
cjclearpool ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type connector -name <J2EEアプリケーション名> -resname <リソースアダプタの表示名>
```

実行例

```
cjclearpool MyServer -type connector -name App1 -resname Rar1
```

cjclearpool コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclearpool（コネクションプール内のコネクション削除）」を参照してください。

参考

Connector 1.5 の仕様に準拠するリソースアダプタの場合，-resname オプションの＜リソースアダプタの表示名＞は次の形式で指定します。

＜リソースアダプタの表示名＞!＜コネクション定義識別子＞

6 リソースアダプタ以外の J2EE リソースの設定

この章では，サーバ管理コマンドを使用した，リソースアダプタ以外の J2EE リソースの設定方法について説明します。

6.1　リソースアダプタ以外の J2EE リソースの設定概要

次の J2EE リソースについて，設定作業の概要を示します。

- JavaBeans リソースの設定
- メールコンフィグレーションの設定
- J2EE リソース共通の設定

6.1.1　JavaBeans リソースの設定の概要

JavaBeans リソースを利用する場合に必要な設定です。

表 6-1　JavaBeans リソースの設定の概要

設定項目	内容	参照先
JavaBeans リソースのインポート	JavaBeans リソースをインポートして，JavaBeans リソース属性の値を設定します。	6.2.1
JavaBeans リソースのプロパティの定義	JavaBeans リソースの実行に必要な情報を設定します。	6.2.2
JavaBeans リソースの開始	JavaBeans リソースを開始します。	6.2.3
JavaBeans リソースの停止	JavaBeans リソースを停止します。	6.2.4
JavaBeans リソースの削除	JavaBeans リソースを削除します。JavaBeans リソースを入れ替える場合は，必要に応じて実行してください。	6.2.5
JavaBeans リソースの状態表示	インポートされているすべての JavaBeans リソースの状態を表示します。	6.2.6

6.1.2　メールコンフィグレーションの設定の概要

JavaMail を使用して SMTP サーバに接続する場合に必要な設定です。メールサーバおよびメールのヘッダに入れる情報を設定します。

表 6-2　メールコンフィグレーションの設定の概要

設定項目	内容	参照先
メールコンフィグレーションの新規作成	メールコンフィグレーションを新規作成します。	6.3.1
メールコンフィグレーションのプロパティ定義	メールコンフィグレーションの次の情報を設定します。 • SMTP メールサーバのホスト名または IP アドレス • メールのヘッダに入れる情報	6.3.2

設定項目	内容	参照先
メールコンフィグレーションの接続テスト	メールコンフィグレーションに設定した内容が正しいかどうかを検証します。	6.3.3
メールコンフィグレーションの削除	作成したメールコンフィグレーションを削除します。	6.3.4
メールコンフィグレーションのコピー	メールコンフィグレーションのプロパティをコピーします。	6.3.5

6.1.3　J2EE リソース共通の設定の概要

J2EE リソースに共通の設定です。

表 6-3　J2EE リソースに共通な設定の概要

設定項目	内容	参照先
J2EE リソース一覧の参照	次の J2EE リソースについて一覧を参照します。 • EJB-JAR ファイルに含まれる J2EE リソース • WAR ファイルに含まれる J2EE リソース • メールコンフィグレーション	6.4
JNDI 名前空間に登録される J2EE リソース名の参照と変更	J2EE リソースの JNDI 名前空間への登録名を参照して，必要に応じて別名を付与します。	6.5

6.2 JavaBeans リソースの設定

JavaBeans リソースの設定とは JavaBeans リソースを利用できる状態にすることです。

JavaBeans リソースの設定をするには，あらかじめ，JavaBeans リソース属性ファイルのテンプレートファイルを参考に，JavaBeans リソース属性ファイルを編集しておいてください。JavaBeans リソース属性ファイルのテンプレートファイル（jb_template.xml）は，次のディレクトリに格納されています。

Windows の場合

```
＜Application Serverのインストールディレクトリ＞¥CC¥admin¥templates¥
```

UNIX の場合

```
/opt/Cosminexus/CC/admin/templates/
```

JavaBeans リソースは，次の手順で設定します。

JavaBeans リソースを新規に設定する場合

1. JavaBeans リソースをインポートします。
2. JavaBeans リソースを開始します。

JavaBeans リソースのプロパティ属性を変更する場合

1. JavaBeans リソースを停止します。
2. JavaBeans リソースのプロパティ定義を変更します。
3. JavaBeans リソースを開始します。

JavaBeans リソースを入れ替える場合

1. JavaBeans リソースを停止します。
2. J2EE サーバを再起動します。
3. JavaBeans リソースを削除します。
4. JavaBeans リソースをインポートします。
5. JavaBeans リソースを開始します。

6.2.1 JavaBeans リソースのインポート

次に示すコマンドを実行して JavaBeans リソースをインポートします。

実行形式

```
cjimportjb ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -f ＜JARファイルパス＞ -c
 ＜JavaBeansリソース属性ファイルパス＞
```

実行例

```
cjimportjb MyServer -f Myjavabeans.jar -c Myjavabeansprop.xml
```

JavaBeans リソースは，cjimportjb コマンドの-d オプションで，ディレクトリパスを指定すると，次のようにインポートすることもできます。

- アーカイブの作成をしないで，ディレクトリ構成でインポートします。
- 指定したディレクトリ下のすべてのアーカイブされた JavaBeans リソースをインポートします。

コマンドの指定方法を次に示します。

実行形式

```
cjimportjb ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -d ＜ディレクトリパス＞ -c ＜JavaBeansリソース属性ファイルパス＞
```

実行例

```
cjimportjb MyServer -d MydirectoryPath -c Myjavabeansprop.xml
```

cjimportjb コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportjb（JavaBeans リソースのインポート)」を参照してください。

注意事項

cjimportjb コマンドの-d オプションには，インポートするディレクトリの最上位を指定してください。また，指定したディレクトリ下にあるすべてのファイルがインポートされるので，不要なファイルは指定したディレクトリ内に含めないようにしてください。

6.2.2 JavaBeans リソースのプロパティ定義

JavaBeans リソースのプロパティを定義します。

プロパティの設定手順の概要については，「3.5 属性ファイルによるプロパティの設定」を参照してください。次に JavaBeans リソースのプロパティ定義について説明します。

(1) 編集する属性ファイル

JavaBeans リソース属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して，JavaBeans リソース属性ファイルを取得します。

実行形式

```
cjgetjbprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜JavaBeans
リソース表示名＞ -c ＜JavaBeansリソース属性ファイルパス＞
```

実行例

```
cjgetjbprop MyServer -resname MyJavaBeansName -c MyJavaBeansProp.xml
```

- 属性の設定

次に示すコマンドを実行して，JavaBeans リソース属性ファイルの値を反映します。

実行形式

```
cjsetjbprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜JavaBeans
リソース表示名＞ -c ＜JavaBeansリソース属性ファイルパス＞
```

実行例

```
cjsetjbprop MyServer -resname MyJavaBeansName -c MyJavaBeansProp.xml
```

(3) 編集する属性設定項目

JavaBeans リソースのプロパティ設定項目を次に示します。

- 実行時プロパティ
- 別名情報

(a) 実行時プロパティ

JavaBeans リソースの実行時プロパティ（<property>タグ）の設定項目を次に示します。

項目	対応するタグ
プロパティ名	<property-name>
プロパティのデータ型	<property-type>
プロパティの値	<property-value>

定義するプロパティの数だけ，上記の設定を繰り返してください。

プロパティ名（<property-name>）の設定項目に，JavaBeans リソースの set メソッドや get メソッドのメソッド名を指定します。プロパティ名（<property-name>）の設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.1.12 <property>タグに指定できるプロパティ」を参照してください。

(b) 別名情報

JavaBeans リソースの別名情報（<resource-env-external-property>タグ）の別名（<optional-name>）を設定します。

JavaBeans リソースの別名の使用法については，「6.5　JNDI 名前空間に登録される J2EE リソース名の参照と変更」を参照してください。

6.2.3　JavaBeans リソースの開始

次に示すコマンドを実行して JavaBeans リソースを開始します。

実行形式

```
cjstartjb ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <JavaBeansリソース表示名>
```

実行例

```
cjstartjb MyServer -resname javabeansname
```

cjstartjb コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartjb（JavaBeans リソースの開始）」を参照してください。

6.2.4　JavaBeans リソースの停止

次に示すコマンドを実行して JavaBeans リソースを停止します。

実行形式

```
cjstopjb ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -resname <JavaBeansリソース表示名>
```

実行例

```
cjstopjb MyServer -resname javabeansname
```

cjstopjb コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstopjb（JavaBeans リソースの停止）」を参照してください。

注意事項

JavaBeans リソースを停止する前に，停止する JavaBeans リソースを使用しているアプリケーションはすべて停止させてください。

6.2.5　JavaBeans リソースの削除

インポートされている JavaBeans リソースを削除します。

準備

JavaBeans リソースを削除する前に，JavaBeans リソースを停止して J2EE サーバを再起動してください。

次に示すコマンドを実行して JavaBeans リソースを削除します。

実行形式

```
cjdeletejb ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -resname ＜JavaBeansリソース表示名＞
```

実行例

```
cjdeletejb MyServer -resname MyJavaBeans
```

cjdeletejb コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeletejb（JavaBeans リソースの削除)」を参照してください。

6.2.6 JavaBeans リソースの状態表示

すべての JavaBeans リソースの JavaBeans リソース名と状態（開始状態または停止状態）を表示します。

次に示すコマンドを実行して JavaBeans リソースの状態を表示します。

実行形式

```
cjlistjb ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］
```

実行例

```
cjlistjb MyServer
```

cjlistjb コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistjb（JavaBeans リソースの一覧表示)」を参照してください。

6.3　メールコンフィグレーションの設定

JavaMail を使用して SMTP サーバに接続する場合に必要な設定です。

6.3.1　メールコンフィグレーションの新規作成

メールコンフィグレーションを新規作成します。

次に示すコマンドを実行してメール属性ファイルを基にメールコンフィグレーションを新規作成します。

実行形式

```
cjsetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type mail -resname <メールの表示名> -c <メール属性ファイルパス>
```

実行例

```
cjsetresprop MyServer -type mail -resname Mail -c MailProp.xml
```

cjsetresprop コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetresprop（リソースの属性設定)」を参照してください。属性ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.3　メール属性ファイル」を参照してください。

6.3.2　メールコンフィグレーションのプロパティ定義

メールコンフィグレーションのプロパティを定義します。新規作成時に定義した内容を変更できます。

プロパティの設定手順の概要については，「3.5　属性ファイルによるプロパティの設定」を参照してください。次にメールコンフィグレーションのプロパティ定義について説明します。

(1)　編集する属性ファイル

メール属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行してメール属性ファイルを取得します。

  実行形式

  ```
  cjgetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type mail -resname <メールの表示名> -c <メール属性ファイルパス>
  ```

実行例

```
cjgetresprop MyServer -type mail -resname Mail -c MailProp.xml
```

- 属性の設定

  次に示すコマンドを実行して，メール属性ファイルを反映します。

  実行形式

```
cjsetresprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type mail -resname
<メールの表示名> -c <メール属性ファイルパス>
```

  実行例

```
cjsetresprop MyServer -type mail -resname Mail -c MailProp.xml
```

(3) 編集する属性設定項目

メールコンフィグレーションのプロパティ設定項目を次に示します。

項目	対応するタグ
E メール送信者の E メールアドレス	<from>
SMTP メールサーバのホスト名，または IP アドレス	<server>
別名情報※	<resource-external-property>

注※　別名情報（<resource-external-property>）には，次の項目を設定します。

別名情報の項目	必須	対応するタグ
リソースの別名	○	<optional-name>
リソース認証方式	△	<res-auth>
リソース共有の有無	△	<res-sharing-scope>

（凡例）○：必須　△：任意

プロパティの設定項目の説明については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「4.3　メール属性ファイル」を参照してください。

6.3.3 メールコンフィグレーションの接続テスト

メールコンフィグレーションに設定した内容が正しいかどうか，メールサーバのコネクションをチェックします。

次に示すコマンドを実行してメールコンフィグレーションの接続テストを実施します。

実行形式

```
cjtestres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type mail -resname ＜メールコンフィグレーションの表示名＞
```

実行例

```
cjtestres MyServer -type mail -resname Mymail1
```

cjtestres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjtestres（リソースの接続テスト)」を参照してください。

6.3.4 メールコンフィグレーションの削除

次に示すコマンドを実行してメールコンフィグレーションを削除します。

実行形式

```
cjdeleteres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type mail -resname ＜メールコンフィグレーションの表示名＞
```

実行例

```
cjdeleteres MyServer -type mail -resname Mail
```

cjdeleteres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeleteres（リソースの削除)」を参照してください。

6.3.5 メールコンフィグレーションのコピー

次に示すコマンドを実行してメールコンフィグレーションのプロパティをコピーします。

実行形式

```
cjcopyres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type mail -src ＜コピー元のメールコンフィグレーションの表示名＞ -dst ＜コピー先のメールコンフィグレーションの表示名＞
```

実行例

```
cjcopyres MyServer -type mail -src Mail -dst Mail2
```

cjcopyres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjcopyres（リソースのコピー)」を参照してください。

6.4 J2EE リソース一覧の参照

J2EE リソースについて，次の一覧を参照します。

- インポートされている EJB-JAR と，EJB-JAR に含まれる J2EE リソース
- インポートされている WAR と，WAR に含まれる J2EE リソース
- インポートされているメールコンフィグレーション

●EJB-JAR ファイルに含まれる J2EE リソースの一覧の参照

次に示すコマンドを実行して，EJB-JAR ファイルに含まれる J2EE リソースの一覧を参照します。

実行形式

```
cjlistres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type ejb ［-resname <EJB-JARファイルの表示名>］
```

- -resname オプションを指定すると，特定の EJB-JAR ファイルに含まれるリソース（SessionBean，EntityBean，MessageDrivenBean）の一覧が表示されます。
- -resname オプションを指定しないと，インポートされている EJB-JAR ファイルの一覧が表示されます。

実行例

```
cjlistres MyServer -type ejb
```

●WAR ファイルに含まれる J2EE リソースの一覧の参照

次に示すコマンドを実行して，WAR ファイルに含まれる J2EE リソースの一覧を参照します。

実行形式

```
cjlistres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type war ［-resname <WARファイルの表示名>］
```

- -resname オプションを指定すると，特定の WAR ファイルに含まれるリソース（サーブレット，JSP およびフィルタ）の一覧が表示されます。
- -resname オプションを指定しないと，インポートされている WAR ファイルの一覧が表示されます。

実行例

```
cjlistres MyServer -type war
```

●メールコンフィグレーションの一覧の参照

次に示すコマンドを実行して，インポートされているメールコンフィグレーションの一覧を参照します。

実行形式

```
cjlistres ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -type mail
```

実行例

```
cjlistres MyServer -type mail
```

cjlistres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistres（リソースの一覧表示)」を参照してください。

6.5　JNDI 名前空間に登録される J2EE リソース名の参照と変更

JNDI 名前空間に登録される J2EE リソース名を変更します。

J2EE リソース名には別名も付けられます。別名を付けることで，JNDI 名前空間から任意の名前で J2EE リソースを参照できるようになります。なお，この機能を**ユーザ指定名前空間機能**といいます。

なお，J2EE リソースの JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。ユーザ指定名前空間機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.6　Enterprise Bean または J2EE リソースへの別名付与（ユーザ指定名前空間機能)」を参照してください。

別名は，次の属性ファイルに追加できます。

属性ファイル	対応するタグ名
メール属性ファイル	<resource-external-property>
JavaBeans リソース属性ファイル	<resource-env-external-property>

メール属性ファイルの<resource-external-property>タグの編集については，「6.3.2　メールコンフィグレーションのプロパティ定義」を，JavaBeans リソース属性ファイルの<resource-env-external-property>タグの編集については，「6.2.2　JavaBeans リソースのプロパティ定義」の別名情報の設定を参照してください。

注意事項

開始状態の J2EE アプリケーションがある場合，別名を設定している J2EE リソースの停止および削除はできません。J2EE サーバで開始されているすべての J2EE アプリケーションを停止させてください。

7

J2EE アプリケーションの作成

この章では，サーバ管理コマンドを使用した J2EE アプリケーションの作成方法について説明します。

7.1　J2EE アプリケーションの作成の概要

J2EE アプリケーションの作成とは，アプリケーション開発環境で作成した Enterprise Bean（EJB-JAR）およびサーブレットと JSP（WAR）を，一つの J2EE アプリケーション（EAR）にすることです。また，J2EE アプリケーションに，その J2EE アプリケーションで使用するリソースアダプタを含めることもできます。

J2EE アプリケーションの作成に必要な作業の概要を次の表に示します。

表 7-1　J2EE アプリケーションの作成で実施する作業

実施項目	内容	参照先
Enterprise Bean（EJB-JAR）のインポート	J2EE アプリケーションの構成に Enterprise Bean（EJB-JAR）が含まれる場合に必要な作業です。 J2EE サーバに Enterprise Bean をインポートします。	7.2.1
サーブレットと JSP（WAR）のインポート	J2EE アプリケーションの構成にサーブレット・JSP（WAR）が含まれる場合に必要な作業です。 J2EE サーバにサーブレット・JSP をインポートします。	7.2.2
リソースアダプタ（RAR）のインポート	J2EE アプリケーションにリソースアダプタ（RAR）を含める場合，含めるリソースアダプタを J2EE サーバにインポートします。	7.2.3
J2EE アプリケーションの新規作成	インポートした EJB-JAR，WAR を基に EAR を作成します。また，必要に応じて，J2EE アプリケーションに RAR を含めます。	7.2.4
J2EE アプリケーションへのライブラリ JAR ファイルの追加	J2EE アプリケーションにライブラリ JAR（ファイル拡張子は小文字の「.jar」）を追加します。	7.2.5
ライブラリ JAR ファイルの一覧の参照	J2EE アプリケーションに含まれているライブラリ JAR ファイルの一覧を参照します。	7.2.6
J2EE アプリケーションからのライブラリ JAR ファイルの削除	J2EE アプリケーションに含まれているライブラリ JAR ファイルを削除します。	7.2.7
J2EE アプリケーションへの参照ライブラリの設定	J2EE アプリケーションに参照ライブラリを設定します。	7.2.8
J2EE アプリケーションのプロパティの設定	作成した J2EE アプリケーションのプロパティを設定します。	9.
J2EE アプリケーションの実行	設定が完了した J2EE アプリケーションを実行します。また，運用に応じて，停止します。	10.

cosminexus.xml を使用した J2EE アプリケーションを作成する場合は，あらかじめ作成した cosminexus.xml を，EJB-JAR，WAR などと一緒に J2EE アプリケーションに含めて使用してください。

ライブラリ JAR と参照ライブラリは，J2EE アプリケーション内の各モジュールから参照できる共通ライブラリです。

ライブラリ JAR と参照ライブラリの特長を次に示します。

- ライブラリ JAR
  - J2EE アプリケーションに JAR（ファイル拡張子は小文字の「.jar」）を含めます。
  - 複数の J2EE アプリケーションで使用する場合，それぞれの J2EE アプリケーションに追加する必要があります。
  - class ファイルを含めることはできません。必ず JAR ファイル形式にまとめて使用します。
- 参照ライブラリ
  - 参照するライブラリファイルの絶対パスを指定します。
  - 複数の J2EE アプリケーションで使用する場合，同じ参照先を指定するだけで利用できます。
  - JAR ファイル，class ファイルどちらも扱うことができます。

ejb-jar.xml，または web.xml に記載したクラス，メソッド（引数，戻り値，例外），アノテーション情報を読み込むためにロードするクラス※，およびそれらの参照解決に必要なクラスを，J2EE アプリケーション内の各モジュールから参照できる共通ライブラリに含める場合は，ライブラリ JAR を使用する必要があります。これらのクラスを参照ライブラリとして使用した場合，アプリケーションのインポート時に java.lang.NoClassDefFoundError や java.lang.ClassNotFoundException が発生し，アノテーション情報の取得に失敗（KDJE42380-W が出力），または，インポートに失敗します。

注※

アノテーション情報を読み込むためにロードするクラスの詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「17.3　ロード対象のクラスとロード時に必要なクラスパス」を参照してください。

7.2 J2EE アプリケーションの作成の詳細

7.2.1 Enterprise Bean（EJB-JAR）のインポート

EJB コンポーネントを利用して J2EE アプリケーションを実行する前に，Enterprise Bean（EJB-JAR）をインポートします。インポートできるのは，EJB 1.1，2.0，2.1，または 3.0 の仕様に準拠した DD を持つ EJB-JAR，またはアノテーションを使用した EJB-JAR です。

なお，次のディレクトリに EJB の各種サンプルプログラムが格納されています。

Windows の場合

＜ Application Server のインストールディレクトリ＞¥CC¥examples¥

UNIX の場合

/opt/Cosminexus/CC/examples/

DD を持った EJB-JAR の場合，DD が DTD に従っていることを確認してから，インポートを実行してください。DD が DTD の仕様に従っていない EJB-JAR をインポートした場合，メッセージが出力されます。メッセージが出力された場合は，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。

次に示すコマンドを実行して EJB-JAR をインポートします。

実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type ejb -f ＜EJB-JAR ファイルパス＞
```

実行例

```
cjimportres MyServer -type ejb -f account.jar
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

注意事項

- DD のバージョンが 1.1 の EJB-JAR をインポートすると，J2EE サーバ内で DD のバージョンが 2.0 に変更されます。
- EJB 1.1，2.0，2.1，または 3.0 の仕様に準拠しない DD を持つ EJB-JAR はインポートできません。
- 「hitachi-runtime.jar」の名称の EJB-JAR はインポートしないでください。
- インポート時に指定した JAR ファイル名は作業ディレクトリ中のディレクトリ名として使用されます。また，JAR ファイル中に，ホームインタフェースなど，java.rmi.Remote を実装したクラスがあると，usrconf.properties ファイルの定義内容に応じて，デプロイ・スタート時に作業ディレクトリ内にスタブが生成されます。デフォルトの設定の場合，スタブは，ejb-jar.xml の<remote>お

よび<home>に記述されたインタフェースに対して作成されます。usrconf.properties ファイルの ejbserver.deploy.stub.generation.scope キーに app を指定した場合は，JAR ファイルに含まれるリモートインタフェースを実装したすべてのインタフェースのスタブが生成されます。

これらのことを考慮して，作業ディレクトリのパス長がプラットフォームの上限に達しないように JAR ファイル名および java.rmi.Remote を実装したクラスのパッケージ名やクラス名を指定してください。

作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。

7.2.2　サーブレットと JSP（WAR）のインポート

J2EE サーバモードで Web コンポーネントを利用して J2EE アプリケーションを実行する前に，サーブレットおよび JSP（WAR）をインポートします。

なお，インポートできるのは，Servlet 標準仕様に準拠した DD を持つ WAR，またはアノテーションを指定した WAR です。このため，DD が DTD に従っていることを確認してから，インポートを実行してください。DD が DTD の仕様に従っていない WAR をインポートした場合，メッセージが出力されます。メッセージが出力された場合は，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。

次に示すコマンドを実行して WAR をインポートします。

実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type war -f ＜WARファイルパス＞
```

実行例

```
cjimportres MyServer -type war -f account.war
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート)」を参照してください。

注意事項

- DD のバージョンが 2.2 の WAR をインポートすると J2EE サーバ内で DD のバージョンが 2.3 に変更されます。
- Servlet 標準仕様に準拠しない DD を持つ WAR はインポートできません。
- インポート時に指定した WAR ファイル名は作業ディレクトリ中のディレクトリ名として使用されます。また，WAR ファイル内のファイルは，デプロイ・スタート時に作業ディレクトリ内に展開されます。WAR ファイル内に，ホームインタフェースなど，java.rmi.Remote を実装したクラスがあると，usrconf.properties ファイルの定義内容に応じて，デプロイ・スタート時に作業ディレクトリ内にスタブが生成されます。usrconf.properties ファイルの

ejbserver.deploy.stub.generation.scope キーに app を指定した場合は，WAR ファイルに含まれるリモートインタフェースを実装したすべてのインタフェースのスタブが生成されます。

これらのことを考慮して，作業ディレクトリのパス長がプラットフォームの上限に達しないように WAR ファイル名，WAR ファイル中のパッケージ名やクラス名および java.rmi.Remote を実装したクラスのパッケージ名やクラス名を指定してください。

作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。

- 「hitachi-runtime.jar」の名称の WAR はインポートしないでください。

7.2.3　リソースアダプタ（RAR）のインポート

J2EE アプリケーションに含めるリソースアダプタ（RAR）をインポートします。J2EE アプリケーションにリソースアダプタを含めると，同じ J2EE アプリケーションの Enterprise Bean（EJB-JAR），サーブレットおよび JSP（WAR）からリソースアダプタが使用できます。

インポートできるリソースアダプタについては，「5.1.1　利用できるリソースアダプタ」を参照してください。

次に示すコマンドを実行して RAR をインポートします。

実行形式

```
cjimportres ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -f ＜RARファイルパス＞
```

実行例

```
cjimportres MyServer -type rar -f account.rar
```

cjimportres コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportres（リソースのインポート）」を参照してください。

注意事項

Connector 1.0 以降の仕様に準拠する DD を持つ RAR であることを確認してから，インポートを実行してください。J2EE Connector 1.0 以降の仕様に準拠しない DD を持つ RAR はインポートできません。

7.2.4　J2EE アプリケーションの新規作成

インポートした EJB-JAR，WAR を基に J2EE アプリケーションを作成します。また，EJB-JAR，WAR で使用する RAR を追加します。

(1) EJB-JAR ファイルの追加

次に示すコマンドを実行して EJB-JAR ファイルを追加します。J2EE アプリケーションがない場合は，J2EE アプリケーションが新規作成されます。

実行形式

```
cjaddapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type ejb -name ＜J2EEアプリケーション名＞ -resname ＜EJB-JAR表示名＞
```

実行例

```
cjaddapp MyServer -type ejb -name adder -resname adder
```

cjaddapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjaddapp（リソースの追加)」を参照してください。

(2) WAR ファイルの追加

次に示すコマンドを実行して WAR ファイルを追加します。J2EE アプリケーションがない場合は，J2EE アプリケーションが新規作成されます。

実行形式

```
cjaddapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type war -name ＜J2EEアプリケーション名＞ -resname ＜WAR表示名＞
```

実行例

```
cjaddapp MyServer -type war -name adder -resname adder_war
```

cjaddapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjaddapp（リソースの追加)」を参照してください。

(3) RAR ファイルの追加

次に示すコマンドを実行して RAR ファイルを追加します。J2EE アプリケーションがない場合は，J2EE アプリケーションが新規作成されます。

実行形式

```
cjaddapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type rar -name ＜J2EEアプリケーション名＞ -resname ＜RAR表示名＞
```

実行例

```
cjaddapp MyServer -type rar -name adder -resname account-rar
```

cjaddapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjaddapp（リソースの追加)」を参照してください。

(4) 注意事項

- J2EE アプリケーション作成時には，EJB ホームオブジェクトの JNDI 名前空間（HITACHI_EJB/SERVERS/＜サーバ名称＞/EJB/＜ J2EE APP 名称＞/＜ Enterprise Bean 名称＞）の＜ J2EE APP 名称＞に該当する名前が自動的に割り当てられます。JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。
- 展開ディレクトリ形式のアプリケーションに対しては，EJB-JAR ファイル，WAR ファイルおよび RAR ファイルの追加はできません。
- 拡張子が".jar"ではない EJB-JAR ファイル，拡張子が".war"ではない WAR ファイル，および拡張子が".rar"ではない RAR ファイルは，application.xml を省略したアプリケーションに追加できません。
- フィルタは，web.xml を省略した WAR ファイルに追加できません。
- J2EE アプリケーション名には，半角英数字（0～9，A～Z，a～z），または次の特殊文字だけを指定してください。
  プラス（+），ハイフン（-），ピリオド（.），キャレット（^），アンダースコア（_）
- J2EE アプリケーションを作成したあとでは，J2EE アプリケーション名を変更できません。

7.2.5　J2EE アプリケーションへのライブラリ JAR ファイルの追加

J2EE アプリケーションにライブラリ JAR ファイルを追加します。

次に示すコマンドを実行して J2EE アプリケーションへライブラリ JAR ファイルを追加します。

実行形式

```
cjimportlibjar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -f ＜ライブラリJAR名＞
```

実行例

```
cjimportlibjar MyServer -name App1 -f applib.jar
```

cjimportlibjar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportlibjar（ライブラリ JAR のインポート)」を参照してください。

注意事項

- インポート時に指定した JAR ファイル名は，作業ディレクトリ中のディレクトリ名として使用されます。作業ディレクトリのパス長がプラットフォームの上限に達しないように JAR ファイル名を指定してください。作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。
- ライブラリ JAR ファイルの拡張子には小文字の「.jar」を指定してください。

- 展開ディレクトリ形式の J2EE アプリケーションへのライブラリ JAR の追加・削除はできません。
- 「hitachi-runtime.jar」の名称のライブラリ JAR はインポートしないでください。

7.2.6　ライブラリ JAR ファイルの一覧の参照

J2EE アプリケーションにインポートされているライブラリ JAR ファイルの一覧を参照します。

次に示すコマンドを実行して J2EE アプリケーションにインポートされているライブラリ JAR ファイルの一覧を参照します。

実行形式

```
cjlistlibjar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞
```

実行例

```
cjlistlibjar MyServer -name App1
```

cjlistlibjar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistlibjar（ライブラリ JAR の一覧表示)」を参照してください。

7.2.7　J2EE アプリケーションからのライブラリ JAR ファイルの削除

J2EE アプリケーションにインポートされているライブラリ JAR ファイルを削除します。

次に示すコマンドを実行して J2EE アプリケーションからライブラリ JAR ファイルを削除します。

実行形式

```
cjdeletelibjar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -f ＜ライブラリJARファイル名称＞
```

実行例

```
cjdeletelibjar MyServer -name App1 -f applib.jar
```

cjdeletelibjar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeletelibjar（ライブラリ JAR の削除)」を参照してください。

注意事項

展開ディレクトリ形式の J2EE アプリケーションへのライブラリ JAR の追加・削除はできません。

7.2.8　J2EE アプリケーションへの参照ライブラリの設定

J2EE アプリケーションに参照ライブラリを設定します。

J2EE アプリケーションへの参照ライブラリの設定は，J2EE アプリケーション属性ファイルで設定します。プロパティを設定する手順については，「3.5　属性ファイルによるプロパティの設定」を参照してください。

(1)　編集する属性ファイル

アプリケーション属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行してアプリケーション属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name App1 -c C:¥home¥app1.xml
  ```

- 属性の設定

  次に示すコマンドを実行してアプリケーション属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name App1 -c C:¥home¥app1.xml
  ```

(3)　編集する属性設定項目

参照するライブラリ JAR の指定（<ref-libraries>）のクラスパス（<classpath>）を設定します。

ここで設定した参照ライブラリのクラスを属性ファイルの中で指定する場合，サーバ管理コマンドを実行するマシン上にその参照ライブラリファイルをコピーし，サーバ管理コマンド用オプション定義ファイルの USRCONF_JVM_CLASSPATH に，コピーした参照ライブラリファイルのパスを設定する必要があります。

8 J2EE アプリケーションのインポートとエクスポート

この章では，サーバ管理コマンドを使用した J2EE アプリケーションのインポートとエクスポートの方法について説明します。

8.1　J2EE アプリケーションのインポート

J2EE アプリケーションをインポートします。インポートできるのは，DD を持つ J2EE アプリケーション，またはアノテーションを使用した J2EE アプリケーションです。DD を持った J2EE アプリケーションの場合，EAR，EJB-JAR，WAR，および RAR の DD が DTD に従っていることを確認してから，インポートを実行してください。

インポートする J2EE アプリケーションには，次の形式があります。

- アーカイブ形式の J2EE アプリケーション

  J2EE サーバの作業ディレクトリ以下に，EJB-JAR/WAR/RAR ファイルのコンポーネントを持つ J2EE アプリケーションです。
- 展開ディレクトリ形式の J2EE アプリケーション

  J2EE サーバの外部に置かれている構成ファイルを，論理的な J2EE アプリケーションとして使用する J2EE アプリケーションです。また，リロードによる J2EE アプリケーションの入れ替えを実施する場合は，この形式でのインポートが必要になります。J2EE アプリケーションの入れ替えについては，「10.5　J2EE アプリケーションの入れ替え」を参照してください。

8.1.1　アーカイブ形式の J2EE アプリケーションのインポート

次に示すコマンドを実行して，アーカイブ形式の J2EE アプリケーションをインポートします。

実行形式

```
cjimportapp ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -f <EARファイルのパス>
```

実行例

```
cjimportapp MyServer -f C:¥home¥bank.ear
```

cjimportapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

(1)　実行時情報に関する注意事項

実行時情報はアプリケーションサーバの J2EE サーバ独自の情報です。次の情報が含まれています。

表 8-1　実行時情報一覧

項目	情報
J2EE アプリケーションの状態	開始／停止状態の情報
J2EE アプリケーションの各種設定	• Enterprise Bean の実行時プロパティ • 各種リファレンスと J2EE リソースのマッピング

項目	情報
	• CMP フィールドのデータベースのマッピング など
自動生成実装クラス	• EJB ホームオブジェクト • EJB オブジェクト
リモートインタフェースの stub クラス，tie クラス	• EJB ホームオブジェクト • EJB オブジェクト

実行時情報付き EAR ファイルをインポートした場合の注意事項を，次に説明します。

- エクスポートされたときの J2EE アプリケーションの状態が復元されます。したがって，開始済みの J2EE アプリケーションをインポートした場合，すぐに J2EE アプリケーションが開始されます。
- J2EE アプリケーションの各種設定が復元されます。インポートする J2EE サーバ上に，エクスポートした J2EE サーバと同じ名称で J2EE リソースアダプタ，JDBC データソース，または JavaMail コンフィグレーションを作成しておくと，インポート時に参照が解決されます。このため，開始済みの J2EE アプリケーションをインポートする場合は，エクスポートした J2EE サーバと同じ名称で J2EE リソースアダプタ，JDBC データソースまたは JavaMail コンフィグレーションを事前に作成しておいてください。また，J2EE リソースアダプタを開始させた状態で，J2EE アプリケーションをインポートしてください。
- 実行時情報付き EAR ファイルに含まれる，自動生成実行クラス，リモートインタフェースの stub クラス，tie クラスは再利用されます。ただし，旧バージョンでエクスポートした EAR ファイルをインポートした場合は，再作成されます。
- 実行環境情報ファイルは変更しないようにしてください。

(2) EAR の DD に関する注意事項

Java EE の仕様に準拠しない DD を持つ EAR はインポートできません。

DD のバージョンが古い場合，バージョンを次のように変更します。

表 8-2 DD のバージョンの変更

DD	インポート前のバージョン	インポート後のバージョン
application.xml	1.2	1.4
	1.3	
ejb-jar.xml	1.1	2.0
web.xml	2.2	2.3

(3) そのほかの注意事項

- インポートするファイルが，J2EE サーバからエクスポートした J2EE アプリケーションでない場合，次の名称のファイルを含めないでください。
  - hitachi-runtime.jar
  - rar.properties
  - fileinfo.properties
  - META-INF/hitachi-ra.xml
- J2EE アプリケーション開始時には，EJB ホームオブジェクトの JNDI 名前空間（HITACHI_EJB/SERVERS/＜サーバ名称＞/EJB/＜ J2EE APP 名称＞/＜ Enterprise Bean 名称＞）の＜ J2EE APP 名称＞に該当する名前が自動的に割り当てられます。JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。
- EAR ファイル中の EJB-JAR ファイル名は作業ディレクトリ中のディレクトリ名として使用されます。また，EJB-JAR ファイル中に，ホームインタフェースなど，java.rmi.Remote を実装したクラスがあると，usrconf.properties ファイルの定義内容に応じて，デプロイ・スタート時に作業ディレクトリ内にスタブが生成されます。デフォルトの設定の場合，スタブは，ejb-jar.xml の<remote>および<home>に記述されたインタフェースに対して作成されます。usrconf.properties ファイルの ejbserver.deploy.stub.generation.scope キーに app を指定した場合は，EJB-JAR ファイルに含まれるリモートインタフェースを実装したすべてのインタフェースのスタブが生成されます。

  これらのことを考慮して，作業ディレクトリのパス長がプラットフォームの上限に達しないように EJB-JAR ファイル名，java.rmi.Remote を実装したクラスのパッケージ名，およびクラス名を指定してください。

  なお，作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1　J2EE サーバの作業ディレクトリ」を参照してください。
- EAR ファイル中の WAR ファイル名は作業ディレクトリ中のディレクトリ名として使用されます。また，WAR ファイル内のファイルは作業ディレクトリ内に展開されます。WAR ファイル中に，ホームインタフェースなど，java.rmi.Remote を実装したクラスがあると，usrconf.properties ファイルの定義内容に応じて，デプロイ・スタート時に作業ディレクトリ内にスタブが生成されます。usrconf.properties ファイルの ejbserver.deploy.stub.generation.scope キーに app を指定した場合は，WAR ファイルに含まれるリモートインタフェースを実装したすべてのインタフェースのスタブが生成されます。

  これらのことを考慮して，作業ディレクトリのパス長がプラットフォームの上限に達しないように WAR ファイル名，WAR ファイル中のパッケージ名やクラス名，および java.rmi.Remote を実装したクラスのパッケージ名やクラス名を指定してください。
- EAR ファイル中のライブラリ JAR ファイル名は，作業ディレクトリ中のディレクトリ名として使用されます。作業ディレクトリのパス長がプラットフォームの上限に達しないように JAR ファイル名を指定してください。

- EAR ファイル中の EJB-JAR 同士，WAR 同士で表示名が重複している場合はインポートできません。表示名が空文字または指定されていない場合は，ファイル名を変換した文字列が指定されます。この値が重複した場合もインポートできません。
- インポートした実行時情報付きの J2EE アプリケーションの実行時情報にエラーがある場合，J2EE アプリケーションのインポートに失敗します。このとき，J2EE アプリケーションは自動開始されないで削除されます。J2EE アプリケーションが削除されないようにするには，cjimportapp コマンドの-nodelete オプションを指定して J2EE アプリケーションをインポートしてください。このとき，J2EE アプリケーションは削除されないでインポートできますが，自動開始されないので，必要に応じてエラーを取り除き，J2EE アプリケーションを開始してください。
- インポートするアプリケーションに application.xml がある場合の表示名，およびディレクトリ名の変換規則は次のとおりです。
  - application.xml の<display-name>タグの値が表示名となります。
  - <display-name>タグの値がない場合，-f オプションに指定した EAR ファイル名を変換した文字列が表示名になります。

    なお，ファイル名に汎用文字と数字以外の文字が含まれている場合，その文字はアンダースコア(_)に置き換えられます。

    置換対象の文字が続けて出現する場合は一つのアンダースコア(_)に纏められます。
  - application.xml の<module>タグに記述されたパス名から拡張子を除いた名称が EJB-JAR ディレクトリ名，または WAR ディレクトリ名となります。
- インポートするアプリケーションに application.xml がない場合の表示名，およびディレクトリ名の変換規則は次のとおりです。
  - -f オプションに指定したファイル名から拡張子を除き，変換した文字列が表示名になります。

    なお，半角英数字（0～9，A～Z，a～z），アンダースコア(_)以外の文字はアンダースコア(_)に置き換えられます。
  - ピリオド（.）が最初に現れるファイル名の場合は，ピリオド（.）を除かないで変換した文字列が表示名になります。
  - EJB-JAR ディレクトリ名の最後は「_jar」，WAR ディレクトリ名の最後は「_war」となります。

8.1.2 展開ディレクトリ形式の J2EE アプリケーションのインポート

展開ディレクトリ形式の J2EE アプリケーションは，J2EE サーバの作業ディレクトリ以下に各コンポーネントの class ファイルや JAR ファイルを持ちません。DD と展開ディレクトリのパス情報だけを持ち，EAR ファイルまたは ZIP ファイルが展開ディレクトリのパスに展開されている J2EE アプリケーションです。ただし，J2EE アプリケーションに含まれる RAR ファイルはアーカイブ形式で格納されています。

展開ディレクトリ形式の J2EE アプリケーションのインポートには，次の二つの方法があります。

- アプリケーションディレクトリを展開ディレクトリ形式のアプリケーションとしてインポートします。

- J2EE アプリケーションを展開ディレクトリ形式のアプリケーションとしてインポートします。

> **注意事項**
>
> 展開ディレクトリ形式のアプリケーションをインポートした場合，J2EE サーバの起動処理中またはサーバ管理コマンド実行中に，アプリケーションディレクトリ以下のディレクトリおよびファイルの追加，削除および上書きはしないでください。

展開ディレクトリ形式のアプリケーションのインポート手順を，次に示します。

(1) アプリケーションディレクトリを展開ディレクトリ形式のアプリケーションとしてインポート

次の手順で，J2EE サーバにアプリケーションディレクトリを登録します。

1. アプリケーションディレクトリを作成します。

   展開ディレクトリ形式の J2EE アプリケーションを新規に作成する場合は，アプリケーションディレクトリを作成します。アプリケーションディレクトリの構成については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「18.4.2　アプリケーションディレクトリの構成」を参照してください。

2. アプリケーションディレクトリをインポートします。

   上記 1.で作成した，アプリケーションディレクトリを J2EE サーバに登録します。

3. 次に示すコマンドを実行して，アプリケーションディレクトリをインポートします。

(a) 実行形式

```
cjimportapp ［<サーバ名>］ -a <アプリケーションディレクトリパス>
```

(b) 実行例

```
cjimportapp MyServer -a AppDirPath
```

cjimportapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート)」を参照してください。

(c) 注意事項

- リモート環境では実行できません。
- application.xml に<alt-dd>タグを指定している，アプリケーションディレクトリを展開ディレクトリ形式で使用できません。
- インポートするアプリケーションに application.xml がある場合の表示名，およびディレクトリ名の変換規則は次のとおりです。

- application.xml の<display-name>タグの値が表示名となります。
- <display-name>タグの値がない場合，-f オプションに指定した EAR ファイル名を変換した文字列が表示名になります。

  なお，ファイル名に汎用文字と数字以外の文字が含まれている場合，その文字はアンダースコア(_)に置き換えられます。

  置換対象の文字が続けて出現する場合は一つのアンダースコア(_)に纏められます。
- application.xml の<module>タグに記述されたパス名から拡張子を除いた名称が EJB-JAR ディレクトリ名，または WAR ディレクトリ名となります。

- インポートするアプリケーションに application.xml がない場合の表示名，およびディレクトリ名の変換規則は次のとおりです。
  - -f オプションに指定したファイル名から拡張子を除き，変換した文字列が表示名になります。

    なお，半角英数字（0～9，A～Z，a～z），アンダースコア(_)以外の文字はアンダースコア(_)に置き換えられます。
  - ピリオド（.）が最初に現れるファイル名の場合は，ピリオド（.）を除かないで変換した文字列が表示名になります。
  - EJB-JAR ディレクトリ名の最後は「_jar」，WAR ディレクトリ名の最後は「_war」となります。
- J2EE サーバ内で，同じディレクトリをアプリケーションディレクトリとする J2EE アプリケーションがすでにある場合，インポートできません。
- -d オプションを使用して，EAR ファイルまたは ZIP ファイルを展開して作成されたアプリケーションディレクトリが，次の条件のどれかと一致する場合，-a オプションでインポートして展開ディレクトリ形式のアプリケーションディレクトリとして使用できません。
  - EJB-JAR のモジュール名が「.jar」で終わっていない。
  - WAR のモジュール名が「.war」で終わっていない。
  - 拡張子を含まないモジュール名称が，拡張子を含まないほかのモジュール名称と同一である。
  - 拡張子を含まないモジュールの名称が，EAR ファイル内のディレクトリと同一である。
- アプリケーションディレクトリ配下の各種コンポーネントの DD は UTF-8 エンコーディングになります。
- DD のバージョンが古い場合，バージョンを次のように変更します。

表 8-3　DD のバージョンの変更

DD	インポート前のバージョン	インポート後のバージョン
application.xml	1.2	1.4
	1.3	
ejb-jar.xml	1.1	2.0
web.xml	2.2	2.3

- 展開ディレクトリ形式の J2EE アプリケーションをインポートする際に，EJB-JAR ディレクトリと WAR ディレクトリを除くアプリケーションディレクトリ以下を対象にライブラリ JAR を検索します。このため，EJB-JAR ディレクトリと WAR ディレクトリを除くアプリケーションディレクトリ以下に多数のファイルが存在する場合，インポートに時間が掛かることがあります。

(2) J2EE アプリケーションを展開ディレクトリ形式のアプリケーションとしてインポート

EAR ファイルまたはエクスポートした実行時情報付き EAR ファイルを展開ディレクトリ形式でインポートします。

次に示すコマンドを実行して，アプリケーションを展開ディレクトリ形式でインポートします。

(a) 実行形式

```
cjimportapp ［＜サーバ名＞］-f ＜ファイルパス＞ -d ＜展開ディレクトリパス＞
```

EAR ファイルまたは ZIP ファイルの内容は，展開ディレクトリパスに展開されます。

(b) 実行例

```
cjimportapp MyServer -f App1.zip -d ApplicationDir
```

cjimportapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

(c) 注意事項

- J2EE サーバを起動したユーザに対して，展開先ディレクトリパスに指定したディレクトリの書き込み権限が必要です。
- application.xml に<alt-dd>タグを指定している，J2EE アプリケーションを展開ディレクトリ形式で使用することはできません。
- J2EE サーバ内で，同じディレクトリを展開ディレクトリとする J2EE アプリケーションがすでにある場合，インポートできません。
- そのほかの注意事項は，アーカイブ形式の J2EE アプリケーションのインポートと同じです。アーカイブ形式の J2EE アプリケーションのインポートについては，「8.1.1 アーカイブ形式の J2EE アプリケーションのインポート」の注意事項を参照してください。
- アプリケーションディレクトリ配下の各種コンポーネントの DD は UTF-8 エンコーディングになります。

8.2 J2EE アプリケーションのエクスポート

次に示すコマンドを実行して J2EE アプリケーションをエクスポートします。

実行形式

```
cjexportapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -f ＜EARファイルパス＞ ［-raw|-normal］
```

実行時情報を含める場合

-normal を指定してください（デフォルト値）。

実行時情報を含めない場合

-raw を指定してください。

実行例

```
cjexportapp MyServer -name account -f C:¥home¥account.zip
```

この例の場合は，EAR ファイルの実行時情報が含まれます。

cjexportapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjexportapp（J2EE アプリケーションのエクスポート）」を参照してください。

J2EE アプリケーションのエクスポート時の注意事項を，次に示します。

- 実行時情報に関する注意事項
- EAR の DD に関する注意事項

●実行時情報に関する注意事項

- J2EE アプリケーションをエクスポートする場合，次に示すどちらかを選択できます。デフォルトは実行時情報付きファイルの形式（ZIP 形式）になります。

実行時情報付きファイル（ZIP 形式）

実行時情報を含んだ ZIP 形式のファイルです。なお，実行時情報付きファイルは，アプリケーションサーバ独自のファイルです。

EAR

Java 2 Platform, Enterprise Edition で規定された EAR です。実行時情報を含みません。

実行時情報を含んだ形式でエクスポートした J2EE アプリケーションには，下位の互換性はありません。J2EE アプリケーションの内容は，変更しないでください。

複数の J2EE サーバから構成されるシステムで，J2EE サーバ間で JDBC リソースの設定，利用するデータベース，および J2EE アプリケーションの実行時プロパティを同じにする場合は，実行時情報付き EAR ファイルを使用します。このとき一方の J2EE サーバで作成した J2EE アプリケーションをエクスポートし，他方の J2EE サーバ上にインポートさせることを推奨します。

J2EE サーバ上に作成した J2EE アプリケーションを別のシステムの J2EE サーバや同じシステムでも実行環境が異なる J2EE サーバにインポートする場合，またはほかの J2EE サーバ製品でも使用する場合には，EAR ファイル形式を使用して J2EE アプリケーションをエクスポートしてください。

- J2EE サーバに登録されている展開ディレクトリ形式のアプリケーションをエクスポートすると，EAR ファイルまたは実行時情報付き ZIP ファイルを生成します。
- アプリケーション開始時に JSP 事前コンパイルを実行したアプリケーションをエクスポートすると，エクスポートされた EAR ファイルには JSP コンパイル結果が含まれます。JSP コンパイル結果が含まれた EAR ファイルを，ほかの J2EE サーバにインポートすると，EAR ファイルに含まれる JSP ファイルのコンパイル結果を使用できます。
- インポートした J2EE アプリケーションが cosminexus.xml を含んでいる場合，cosminexus.xml を含んだ状態でエクスポートします。cosminexus.xml を含むアプリケーションのエクスポートについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16.3.6　cosminexus.xml を含むアプリケーションの運用」を参照してください。

●EAR の DD に関する注意事項

- EAR の DD のバージョンは，J2EE アプリケーションのバージョンによって，1.4 以降になります。
- EAR に含まれる EJB-JAR の DD のバージョンは，EJB-JAR のバージョンによって，2.0 以降になります。
- EAR に含まれる WAR の DD のバージョンは，WAR のバージョンによって，2.3 以降になります。
- EAR に含まれる各種コンポーネントの DD は UTF-8 エンコーディングになります。

9 J2EE アプリケーションのプロパティ設定

この章では，サーバ管理コマンドを使用した J2EE アプリケーションのプロパティの設定方法について説明します。

9.1　J2EE アプリケーションのプロパティ設定の概要

J2EE アプリケーションのプロパティ設定とは，インポートした J2EE アプリケーションの属性や動作を定義することです。この節では，次の説明をします。

- J2EE アプリケーションのプロパティ設定と属性ファイル
- Enterprise Bean のプロパティ設定項目
- サーブレットと JSP のプロパティ設定項目
- J2EE アプリケーション（共通）のプロパティ設定項目

> **注意事項**
>
> アノテーションで指定した項目は変更できません。また，cosminexus.xml を含むアプリケーションのプロパティを変更するには，サーバ管理コマンドを使って設定する方法と，MyEclipce 使って設定する方法があります。サーバ管理コマンドで cosminexus.xml にある定義情報を変更したあとでリデプロイ機能を使用すると，サーバ管理コマンドで変更した情報は失われます。
>
> MyEclipce を使った cosminexus.xml を含むアプリケーションのプロパティの変更については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.3.2 cosminexus.xml エディタの操作方法」を参照してください。また，cosminexus.xml を含むアプリケーションの運用については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16.3.6　cosminexus.xml を含むアプリケーションの運用」を参照してください。

J2EE アプリケーションに含まれるリソースアダプタのプロパティ設定と属性ファイルについては，「5.4 リソースアダプタのプロパティ定義」を参照してください。

9.1.1　J2EE アプリケーションのプロパティ設定と属性ファイル

プロパティの設定手順の概要については，「3.5　属性ファイルによるプロパティの設定」を参照してください。

次の属性ファイルを利用して，J2EE アプリケーションのプロパティを設定できます。

- 固有の属性ファイルを利用します。
  J2EE アプリケーションを構成するコンポーネントの固有属性ファイルごとに，プロパティを設定します。
- アプリケーション統合属性ファイルを利用します。
  固有の属性ファイルの内容を一括して，プロパティを設定します。

J2EE アプリケーションのプロパティの設定では，固有の属性ファイルとアプリケーション統合属性ファイルを個々に利用することも，合わせて利用することもできます。

この章では，主に，固有の属性ファイルを使用してのプロパティ設定について説明しています。アプリケーション統合属性ファイルを使用してのプロパティ設定については，「9.2　アプリケーション統合属性ファイルによるプロパティ設定」を参照してください。

9.1.2　Enterprise Bean のプロパティ設定項目

Enterprise Bean が含まれる J2EE アプリケーションを作成する場合およびカスタマイズする場合，必要に応じて設定してください。

Enterprise Bean のプロパティ設定項目を次に示します。

表 9-1　Enterprise Bean のプロパティ編集項目

編集項目	内容	必須／任意	固有属性ファイルの編集参照先
Enterprise Bean のリファレンス定義	J2EE アプリケーションを構成する，次の Enterprise Bean のリファレンスを定義します。 • Session Bean • Entity Bean • Message-driven Bean	○	9.3
ほかの Enterprise Bean のリファレンス定義	ほかの Enterprise Bean へのリファレンスについて定義します。	○	9.3.1
メールコンフィグレーションのリファレンス定義	メールサーバへのリファレンスについて定義します。	○	9.3.2
リソースアダプタのリファレンス定義	リソースアダプタへのリファレンスについて定義します。	○	9.3.3
リソース環境のリファレンス定義	リソース環境へのリファレンスを定義します。	○	9.3.4
Message-driven Bean のメッセージ参照定義	Message-driven Bean の Connection Factory と Destination を参照する個所に，実際の Connection Factory と Destination をマッピングします。	○	9.4
トランザクション属性の定義	トランザクション属性について定義します。	○	9.5
Entity Bean の CMP 定義	Entity Bean の永続性管理をコンテナに任せる場合に定義します。	○	9.6
CMP の設定	単一プライマリキーを使用して Entity Bean の永続性管理をコンテナに任せる場合に定義します。 複合プライマリキーを使用して Entity Bean の永続性管理をコンテナに任せる場合に定義します。	○	9.6.1
CMP1.x とデータベースのマッピング	CMP1.x Entity Bean のフィールドをデータベース上の表にマッピングします。	○	9.6.2

編集項目		内容	必須／任意	固有属性ファイルの編集参照先
	CMP2.x とデータベースのマッピング	CMP2.x Entity Bean のフィールドをデータベース上の表にマッピングします。	○	9.6.3
Enterprise Bean の実行時属性の定義		Enterprise Bean の実行時の動作について設定します。必要に応じて設定してください。	◎	9.10
	Stateful Session Bean の実行時プロパティの設定	Stateful Session Bean の実行時の動作についてのプロパティを設定します。	◎	9.10.1
	Stateless Session Bean の実行時プロパティの設定	Stateless Bean の実行時の動作についてのプロパティを設定します。	◎	9.10.2
	Entity Bean の実行時プロパティの設定	Entity Bean の実行時の動作についてのプロパティを設定します。	◎	9.10.3
	Message-driven Bean の実行時プロパティの設定	Message-driven Bean の実行時の動作についてのプロパティを設定します。	◎	9.10.4
インターセプタの設定		インターセプタを使用する場合に定義します。	○	9.17

（凡例）

◎：必須。対応する Enterprise Bean が含まれる J2EE アプリケーションのカスタマイズで必ず設定する。

○：任意。J2EE アプリケーションに応じて設定する。

9.1.3　サーブレットと JSP のプロパティ設定項目

サーブレットおよび JSP を含む J2EE アプリケーションを作成する場合に，必要に応じて設定してください。

サーブレットと JSP のプロパティ設定項目を次に示します。

表 9-2　サーブレットと JSP のプロパティ編集項目

編集項目		内容	必須／任意	固有属性ファイルの編集参照先
サーブレットと JSP のリファレンス定義		Enterprise Bean，セキュリティロール，リソース（データベースまたはメールサーバ）へのリファレンス（リソース参照）について定義します。	○	9.7
	Enterprise Bean のリファレンス定義	Enterprise Bean へのリファレンスについて定義します。	○	9.7.1
	メールコンフィグレーションのリファレンスの定義	メールサーバへのリファレンスについて定義します。	○	9.7.2
	リソースアダプタのリファレンス定義	リソースアダプタへのリファレンスについて定義します。	○	9.7.3

編集項目	内容	必須／任意	固有属性ファイルの編集参照先
リソース環境のリファレンス定義	リソース環境リファレンスを定義します。	○	9.7.4
サーブレットと JSP のマッピング定義	サーブレットと JSP のマッピングを定義します。	○	9.8
フィルタの設定	フィルタを追加し，マッピングを定義します。	○	9.9
フィルタの追加	フィルタを追加します。	○	9.9.1
フィルタのマッピング定義	フィルタへのマッピングを定義します。	○	9.9.2
フィルタの削除	フィルタを削除します。	○	9.9.3
サーブレットと JSP の実行時属性の定義	サーブレットおよび JSP の実行時属性を定義します。	○	9.11
J2EE アプリケーションのコンテキストルート定義	J2EE アプリケーションのコンテキストルートを設定します。	◎	9.11.1
Web アプリケーション単位での同時実行スレッド数制御の定義	Web コンテナで Web アプリケーション単位での同時実行スレッド数を制御するかどうか，およびスレッド数などを設定します。	○	9.11.2
URL グループ単位での同時実行スレッド数制御の定義	URL グループ単位での同時実行スレッド数を制御するための定義をします。	○	9.11.3
URL グループ単位の実行待ちリクエスト数の監視の定義	URL グループ単位の実行待ちリクエスト数を監視するための定義をします。	○	9.11.4
デフォルトの文字エンコーディングの設定	Web アプリケーションのデフォルトの文字エンコーディングを設定します。	○	9.12
サーブレットと JSP のエラー通知の設定	サーブレット，JSP でエラーが発生した場合に，J2EE アプリケーションにエラーを通知するための設定をします。	○	9.16

（凡例）

◎：必須。サーブレットと JSP が含まれる J2EE アプリケーションのカスタマイズでは必ず設定する。

○：任意。J2EE アプリケーションに応じて設定する。

9.1.4　J2EE アプリケーション（共通）のプロパティ設定項目

J2EE アプリケーションの構成に関係なく，必要に応じて設定してください。

表 9-3　J2EE アプリケーション（共通）のプロパティ編集項目

編集項目	内容	必須／任意	固有属性ファイルの編集参照先	
			参照先マニュアル※	参照個所
JNDI 名前空間に登録される名称の参照と変更	J2EE アプリケーションの JNDI 名前空間への登録名を参照して，必要に応じて別名を付与します。	○	このマニュアル	9.13

編集項目		内容	必須／任意	固有属性ファイルの編集参照先	
				参照先マニュアル※	参照個所
	J2EE アプリケーション名の参照	J2EE アプリケーション名を参照するための設定をします。	○	このマニュアル	9.13.1
	Enterprise Bean 名の参照と変更	Enterprise Bean 名を参照するための設定および Enterprise Bean 名の変更をします。	○	このマニュアル	9.13.2
CTM のスケジューリング		CTM を利用してキューのスケジューリングをするかどうかと，スケジューリング方法について設定します。	○	このマニュアル	9.14
	J2EE アプリケーション単位のスケジューリング	J2EE アプリケーション単位の CTM との連携にかかわる設定をします。	○	このマニュアル	9.14.1
	Stateless Session Bean 単位のスケジューリング	Stateless Session Bean 単位のスケジューリングの設定をします。	○	このマニュアル	9.14.2
起動順序の設定		J2EE アプリケーションの起動順序，および J2EE アプリケーションに含まれる Enterprise Bean（EJB-JAR）やサーブレットまたは JSP（WAR）の起動順序を設定します。	○	このマニュアル	9.15
	J2EE アプリケーションの起動順序の設定	J2EE アプリケーションの起動順序を設定します。	○	このマニュアル	9.15.1
	Enterprise Bean の起動順序の設定	J2EE アプリケーションに含まれる Enterprise Bean（EJB-JAR）の起動順序を設定します。	○	このマニュアル	9.15.2
	サーブレットと JSP の起動順序の設定	J2EE アプリケーションに含まれるサーブレットまたは JSP（WAR）の起動順序を設定します。	○	このマニュアル	9.15.3
セキュリティロールの設定		セキュリティロールを使用したユーザ管理をする場合に必要な設定です。 ユーザとロールの登録と対応づけができます。	○	機能解説 セキュリティ管理機能編	9.2
	ユーザの設定	ユーザを登録します。ロールとの対応づけもできます。	○	機能解説 セキュリティ管理機能編	9.2.1
	ロールの設定	ロールを登録します。ユーザとの対応づけもできます。	○	機能解説 セキュリティ管理機能編	9.2.2
セキュリティロールのリファレンス定義		セキュリティロールへのリファレンスについて定義します。	○	機能解説 セキュリティ管理機能編	9.3
	Enterprise Bean のセキュリティロールリファレンスの定義	セキュリティロールを参照している個所に，実際に J2EE サーバが管理しているセキュリティロールをマッピングします。	○	機能解説 セキュリティ管理機能編	9.3.1

編集項目		内容	必須／任意	固有属性ファイルの編集参照先	
				参照先マニュアル※	参照個所
	サーブレットと JSP のセキュリティロールリファレンスの定義	セキュリティロールへのリファレンスについて定義します。	○	機能解説 セキュリティ管理機能編	9.3.2
セキュリティの定義（メソッドパーミッション）		セキュリティロールによるアクセス権を設定する場合のメソッドのパーミッションを定義します。 すべてのユーザに対してアクセス権限を設定する場合のメソッドのパーミッションを定義します。	○	機能解説 セキュリティ管理機能編	9.4
セキュリティの定義（セキュリティアイデンティティ）		セキュリティの情報であるセキュリティアイデンティティとして UseCallerIdentity を設定します。 セキュリティの情報であるセキュリティアイデンティティとして Run as を設定します。	○	機能解説 セキュリティ管理機能編	9.5
	Enterprise Bean のセキュリティアイデンティティ	J2EE アプリケーション内で参照しているセキュリティアイデンティティ情報に，J2EE サーバが実際に管理しているユーザ ID をマッピングします。	○	機能解説 セキュリティ管理機能編	9.5.1
	サーブレットと JSP のセキュリティアイデンティティ	セキュリティアイデンティティを定義します。	○	機能解説 セキュリティ管理機能編	9.5.2

（凡例）

○：任意。J2EE アプリケーションに応じて設定する。

注※

「参照先マニュアル」に示したマニュアル名の「アプリケーションサーバ」は省略しています。

9.2　アプリケーション統合属性ファイルによるプロパティ設定

9.2.1　設定手順

サーバ管理コマンドでアプリケーション統合属性ファイルを取得して，固有の属性ファイルの内容を一括して編集できます。

アプリケーション統合属性ファイルに含まれる，固有の属性ファイルの要素を次に示します。

- アプリケーション属性ファイル
  （<hitachi-application-property></hitachi-application-property>）
- EJB-JAR 属性ファイル
  （<hitachi-ejb-jar-property></hitachi-ejb-jar-property>）
- Session Bean 属性ファイル
  <hitachi-session-bean-property></hitachi-session-bean-property>
- Entity Bean 属性ファイル
  <hitachi-entity-bean-property></hitachi-entity-bean-property>
- Message-driven Bean 属性ファイル
  <hitachi-message-bean-property></hitachi-message-bean-property>
- WAR 属性ファイル
  <hitachi-war-property></hitachi-war-property>
- フィルタ属性ファイル
  <hitachi-filter-property></hitachi-filter-property>
- サーブレット属性ファイル
  <hitachi-servlet-property></hitachi-servlet-property>
- Connector 属性ファイル
  <hitachi-connector-property></hitachi-connector-property>

アプリケーション統合属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.1　アプリケーション統合属性ファイル」を参照してください。

アプリケーション統合属性ファイルによる，J2EE アプリケーションのプロパティ設定手順を次に示します。

(1)　アプリケーション統合属性ファイルの取得

アプリケーション統合属性ファイルを取得するために，アプリケーション統合属性ファイルパスを指定して，cjgetappprop コマンドを実行します。

実行形式

```
cjgetappprop ＜サーバ名称＞ -name ＜J2EEアプリケーション名＞ -type all -c ＜アプリケーション統合属性ファイルパス＞
```

実行例

```
cjgetappprop Myserver -name App1 -type all -c App1prop.xml
```

(2) プロパティ設定項目の編集

上記(1)で出力されたアプリケーション統合属性ファイルをテキストエディタで編集します。J2EE アプリケーションのプロパティ編集項目については，「9.1 J2EE アプリケーションのプロパティ設定の概要」を参照してください。アプリケーション統合属性ファイルの編集項目は，固有の属性ファイルの編集項目と同じです。

(3) アプリケーション属性の設定

アプリケーション統合属性ファイルで指定した項目をアプリケーション属性に反映するために，cjsetappprop コマンドを実行します。

実行形式

```
cjsetappprop ＜サーバ名称＞ -name ＜J2EEアプリケーション名＞ -type all -c ＜アプリケーション統合属性ファイルパス＞
```

実行例

```
cjsetappprop MyServer -name App1 -type all -c App1Prop.xml
```

9.3　Enterprise Bean のリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean のリファレンス（リソース参照）を定義します。Enterprise Bean には，次の種類があります。

- Session Bean
- Entity Bean
- Message-driven Bean

それぞれの Enterprise Bean には，次に示す 3 種類のリファレンスがあります。

- Enterprise Bean リファレンス
  ほかの Enterprise Bean 呼び出しのための参照です。
- リソースリファレンス
  リソースへの参照です。メールリファレンス，リソースアダプタリファレンス，およびリソース環境リファレンスがあります。
- セキュリティロールリファレンス
  Enterprise Bean のメソッド呼び出しのアクセス制御をするために使用するロール名称の参照です。

この節では，Enterprise Bean リファレンスとリソースリファレンスについて説明します。セキュリティロールのリファレンスについては，マニュアル「アプリケーションサーバ 機能解説 セキュリティ管理機能編」の「9.3　セキュリティロールのリファレンス定義」を参照してください。

なお，プロパティの設定項目については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1　MessageDrivenBean 属性ファイルの指定内容」

9.3.1　ほかの Enterprise Bean のリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean が，ほかの Enterprise Bean を呼び出している場合，リファレンスを解決するためのプロパティを設定します。

(1)　編集する属性ファイル

J2EE アプリケーションを構成する Enterprise Bean の種類に対応する属性ファイルを編集します。

- Session Bean 属性ファイル
- Entity Bean 属性ファイル
- Message-driven Bean 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次に示すコマンドを実行して Enterprise Bean の属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Beanの属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder/adder_eb -c C:¥home¥adder_ejb.xml
  ```

- **属性の設定**

  次に示すコマンドを実行して，Enterprise Bean の属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Beanの属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type ejb -resname adder/adder_eb -c C:¥home¥adder_ejb.xml
  ```

(3) 編集する属性設定項目

参照する Enterprise Bean のアクセスタイプには，次の 2 種類があります。

参照する Enterprise Bean のアクセスタイプ	対応するタグ名
リモートインタフェース	<ejb-ref>
ローカルインタフェース	<ejb-local-ref>

注　ビジネスインタフェース（<business-local>および<business-remote>）は，アノテーションで設定しているため変更できません。

それぞれの設定項目について説明します。

(a) リモートインタフェースの Enterprise Bean のリファレンス定義

リモート Enterprise Bean のリファレンス設定項目（<ejb-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
Enterprise Bean 参照名	○	<ejb-ref-name>
参照する Enterprise Bean の種別	○	<ejb-ref-type>
リモートホームインタフェース	○	<home>
リモートインタフェース	○	<remote>
リンク先の<ejb-name>	△	<ejb-link>

（凡例） ○：必須　△：任意

(b) ローカルインタフェースの Enterprise Bean のリファレンス定義

ローカル Enterprise Bean のリファレンス設定項目（<ejb-local-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
Enterprise Bean 参照名	○	<ejb-ref-name>
参照する Enterprise Bean の種別	○	<ejb-ref-type>
ローカルホームインタフェース	○	<local-home>
ローカルインタフェース	○	<local>
リンク先の<ejb-name>	△	<ejb-link>

（凡例） ○：必須　△：任意

(4) 注意事項

- リモートインタフェースの Enterprise Bean を呼び出す場合には，別アプリケーションにある Enterprise Bean をマッピング先として，「リンク先の<ejb-name>」（<ejb-link>）に設定できます。

  マッピング先の Enterprise Bean がアプリケーションサーバ上で動作している場合，ネーミングの切り替え機能を利用して，次に示すような形式でマッピング先を指定できます。

  指定形式

  ```
  corbaname::＜名前空間のホスト名＞:＜名前空間のポート番号＞#＜EJBホームオブジェクトリファレンスのJNDI名＞
  ```

  ＜名前空間のホスト名＞

  　マッピング先 Enterprise Bean が使用する名前空間が動作する，ホスト名称を指定します。

  ＜名前空間のポート番号＞

  　マッピング先 Enterprise Bean が使用する名前空間が動作する，ポート番号を指定します。

＜EJB ホームオブジェクトリファレンスの JNDI 名＞

次に示す形式に従って，JNDI 名を指定します。

```
HITACHI_EJB/SERVERS/＜サーバ名称＞/EJB/＜J2EE APP名称＞/＜Enterprise Bean名称＞
```

＜サーバ名称＞：J2EE サーバのサーバ名称

＜J2EE APP 名称＞：J2EE アプリケーションのルックアップ名称

＜Enterprise Bean 名称＞：Enterprise Bean のルックアップ名称

指定例

```
corbaname::MyHost:900#HITACHI_EJB/SERVERS/MyServer/EJB/MyApplication/MyBean
```

- ネーミングサービスがローカルホスト上で動作する場合，J2EE サーバ用の usrconf.properties ファイルの ejbserver.naming.host キーには，「localhost」の文字列を使用しないで，マシン名または IP アドレスを指定してください。

9.3.2　メールコンフィグレーションのリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean が，メールコンフィグレーションを参照している場合，リファレンスを定義するためのプロパティを設定します。

(1)　編集する属性ファイル

編集する属性ファイルについては，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(1)　編集する属性ファイル」を参照してください。

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの取得と属性の設定については，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

メールコンフィグレーションのリファレンス設定項目（<resource-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
リソース参照名	○	<res-ref-name>
リソースタイプ	○	<res-type>
リソース認証方式	○	<res-auth>
リソース共有可否	△	<res-sharing-scope>
リンク先のメールコンフィグレーション名	△	<linked-to>

（凡例）　○：必須　△：任意

9.3.3　リソースアダプタのリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean が，リソースアダプタを参照している場合，リファレンスを解決するためのプロパティを設定します。

(1)　編集する属性ファイル

編集する属性ファイルについては，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(1)　編集する属性ファイル」を参照してください。

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの取得と属性の設定については，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

リソースアダプタのリファレンス設定項目（<resource-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
リソース参照名	○	<res-ref-name>
リソースタイプ	○	<res-type>
リソース認証方式	○	<res-auth>
リソース共有可否	△	<res-sharing-scope>
リンク先のリソースアダプタ表示名	△	<linked-to>

（凡例）　○：必須　△：任意

9.3.4　リソース環境のリファレンス定義

J2EE アプリケーションを構成する Enterprise Bean のリソース環境のリファレンスを解決するためのプロパティを設定します。

(1)　編集する属性ファイル

編集する属性ファイルについては，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(1)　編集する属性ファイル」を参照してください。

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの取得と属性の設定については，「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

リソース環境のリファレンス設定項目（<resource-env-ref>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
リソース環境変数名	○	<resource-env-ref-name>
リソース環境変数値のタイプ	○	<resource-env-ref-type>
リンク先キュー情報	△	<linked-queue>
リソースアダプタの表示名	△	<resource-adapter>
キュー名	△	<queue>
リンク先の JavaBeans リソース表示名	△	<linked-to>

（凡例）　○：必須　△：任意

9.4 Message-driven Bean のメッセージ参照定義

9.4.1 定義方法

Message-driven Bean のメッセージ参照を定義します。

(1) 編集する属性ファイル

Message-driven Bean 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Message-driven Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Message-driven Bean表示名> -c <Message-driven Bean属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name message -type ejb -resname message/MyMessage -c C:¥home¥MyMessageBean.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Message-driven Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Message-driven Bean表示名> -c <Message-driven Bean属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name message -type ejb -resname message/MyMessage -c C:¥home¥MyMessageBean.xml
  ```

(3) 編集する属性設定項目

メッセージ参照の設定項目（<message-ref>）を次に示します。

項目	必須	対応するタグ名
Connection Factory 名	○	<connection-factory>

項目	必須	対応するタグ名
リソースアダプタ表示名	○	<connction-destination> - <resource-adapter>
キューの表示名※	○	<connction-destination> - <queue>

（凡例） ○：必須

注※　JMS のキューを複数の異なる Message-driven Bean で共有することは推奨しません。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1　MessageDrivenBean 属性ファイルの指定内容」を参照してください。

9.5　トランザクション属性の定義

9.5.1　定義方法

コンテナによるトランザクションの管理方法を定義します。

トランザクション属性は Enterprise Bean 単位，インタフェース単位，メソッド単位にそれぞれ指定できます。指定がない場合は上位の単位で指定されたトランザクション属性が有効になります。

- Enterprise Bean ごとの適用
  割り当てたトランザクション属性を，Enterprise Bean 内のメソッドに適用します。
- インタフェースごとの適用
  割り当てたトランザクション属性を，インタフェース内のメソッドに適用します。
- メソッドごとの適用
  割り当てたトランザクション属性を，一つのメソッドに適用します。

(1)　編集する属性ファイル

次の属性ファイルのうち，トランザクション属性を設定する Enterprise Bean の種類に対応する属性ファイルを編集します。

- Session Bean 属性ファイル
- Entity Bean 属性ファイル
- Message-driven Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**
  次に示すコマンドを実行して Enterprise Bean の属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Beanの属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder/adder_eb -c C:¥home¥adder_ejb.xml
  ```

- **属性の設定**
  次に示すコマンドを実行して，Enterprise Bean の属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type ejb -resname ＜EJB-JAR表示名＞/＜Enterprise Bean表示名＞ -c ＜Enterprise Beanの属性ファイルパス＞
```

実行例

```
cjsetappprop MyServer -name adder -type ejb -resname adder/adder_eb -c C:¥home¥adder_ejb.xml
```

(3) 編集する属性設定項目

コンテナトランザクション属性（<container-transaction>）の設定項目を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
メソッドの説明	△	<method> - <description>
インタフェース種別	△	<method> - <method-intf>
メソッド名	○	<method> - <method-name>
トランザクション属性	○	<trans-attribute>

（凡例） ○：必須　△：任意

プロパティの設定項目については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1　MessageDrivenBean 属性ファイルの指定内容」

トランザクション属性の指定とトランザクション管理の動作を，次に示します。

- NotSupported
  コンテナ管理トランザクション内でメソッドを実行しません。
- Supports
  メソッドのトランザクションの処理は場合によって異なります。トランザクション内およびトランザクション外で正しく実行するメソッドに対してだけ，この属性を使用してください。
- Required

コンテナはメソッドを常にトランザクション内で実行します。

- RequiresNew

  メソッドは，常に新しいトランザクションで実行します。

- Mandatory

  メソッドは，常にクライアントのトランザクションを使用しなければなりません。つまり，クライアントはトランザクション内でメソッドを呼び出さなければなりません。

- Never

  メソッドはトランザクション内で実行できません。つまり，クライアントは，トランザクションの外でメソッドを呼び出さなければなりません。

(4) 注意事項

- Enterprise Bean の種類によって設定できるトランザクション属性が異なります。設定できるトランザクション属性およびトランザクション管理の動作については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。
- Session Bean のプロパティ（<transaction-type>）で，Session Bean が自身のトランザクションを管理するよう指定されている場合，この設定でトランザクション属性を変更できません。
- Message-driven Bean の場合は，onMessage のメソッドだけ設定できます。

9.6 Entity Bean の CMP 定義

Entity Bean の CMP 定義をします。CMP を定義すると，Enterprise Bean である Entity Bean の永続性管理をコンテナに任せることができます。

9.6.1 CMP の設定

(1) 編集する属性ファイル

Entity Bean 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Entity Bean 属性ファイルを取得します。

  実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名> -c <Entity Bean属性ファイルパス>
```

  実行例

```
cjgetappprop MyServer -name adder -type ejb -resname account/MyAccoub -c C:¥home¥adder_ejb.xml
```

- 属性の設定

  次に示すコマンドを実行して，Entity Bean 属性ファイルの値を反映します。

  実行形式

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名>-c <Entity Bean属性ファイルパス>
```

  実行例

```
cjsetappprop MyServer -name adder -type ejb -resname account/MyAccoub -c C:¥home¥adder_ejb.xml
```

(3) 編集する属性設定項目

CMP の定義は，プライマリキーが単一プライマリキーの場合と複合プライマリキーの場合で，設定方法が異なります。

(a)　単一プライマリキーの場合の属性設定

プライマリキーが単一プライマリキーの場合のプロパティ項目を設定します。

永続性管理種別（<persistence-type>）を確認します。CMP Entity Bean の場合は「Container」が設定されています。

単一プライマリキーのプロパティ設定項目を次に示します。

項目	必須	対応するタグ名
プライマリキーのクラス※1	○	<prim-key-class>
リエントラント可否	○	<reentrant>
永続性管理フィールドの説明	△	<cmp-field> - <description>
永続性管理フィールドのフィールド名※2	△	<cmp-field> - <field-name>
プライマリキーフィールドのフィールド名	△	<primkey-field>

（凡例）　○：必須　△：任意

注※1　この Entity Bean のプライマリキーを含むクラスまたはインタフェースを入力します。プライマリキークラスは，java.lang.Object クラス，またはコンテナ管理フィールドと同じクラスもしくはインタフェースでなければなりません。

注※2　通常は不要です。追加する場合はあらかじめ Entity Bean のクラス定義にも追加しておく必要があります。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」を参照してください。

(b)　複合プライマリキーの場合の属性設定

プライマリキーが複合プライマリキーの場合のプロパティ項目を設定します。

永続性管理種別（<persistence-type>）を確認します。CMP Entity Bean の場合は「Container」が設定されています。

複合プライマリキーのプロパティ設定項目を次に示します。

項目	必須	対応するタグ名
プライマリキーのクラス※1	○	<prim-key-class>
リエントラント可否	○	<reentrant>
永続性管理フィールドの説明	△	<cmp-field> - <description>
永続性管理フィールドのフィールド名※2	△	<cmp-field> - <field-name>

（凡例）　○：必須　△：任意

注※1　この Entity Bean のプライマリキーを含むクラスまたはインタフェースを入力します。プライマリキークラスは，java.lang.Object クラス，またはコンテナ管理フィールドと同じクラスもしくはインタフェースでなければなりません。

注※2　通常は不要です。追加する場合はあらかじめ Entity Bean のクラス定義にも追加しておく必要があります。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」を参照してください。

9.6.2　CMP1.x とデータベースのマッピング

CMP1.x Entity Bean のフィールドをデータベース上の表にマッピングします。

(1)　編集する属性ファイル

Entity Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Entity Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名>-c <Entity Bean属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname account/MyAccoub -c C:¥home¥adder_ejb.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Entity Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名>-c <Entity Bean属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type ejb -resname account/MyAccoub -c C:¥home¥adder_ejb.xml
  ```

(3)　編集する属性設定項目

CMP1.x Entity Bean のフィールドをデータベース上の表にマッピングする（<cmp-map>）プロパティ設定項目を次に示します。

項目	必須	対応するタグ名
リソースアダプタの表示名	○	<datasource-name>

項目	必須	対応するタグ名
データベースのカタログ名	△	<catalog-name>
データベースのスキーマ名	△	<schema-name>
データベースのテーブル名	○	<table-name>
データベースへの書き込み許可／禁止	○	<read-only-access>
トランザクション遮断レベル※1	△	<transaction-isolation>
データベース書き込みデータ照合方法	△	<concurrency-protection>
フィールドとテーブルのカラムとのマッピング情報※2	○	<field-impl>
finder メソッドの検索条件※3	○	<finder-impl>

（凡例） ○：必須　△：任意

注※1

使用できるトランザクション遮断レベル（<transaction-isolation>）の値は，データベースおよび JDBC ドライバでサポートされるオプションによって異なります。

注※2

プライマリキーに対してデータベーステーブルの列を設定します。同じデータベーステーブルのフィールドのマッピングにも使用されます。

EntityBean のフィールド名（<field-name>）の値は変更できません。EntityBean のフィールド名（<field-name>）のマッピング先のデータベースの列（<column-name>）を設定してください。

フィールドとテーブルのカラムとのマッピング情報（<field-impl>）は，次の項目で構成されています。

項目	対応するタグ名
EntityBean のフィールド名	<field-name>
テーブルのカラム名	<column-name>

注※3

finder メソッドのメソッド名（<method-name>）の値は変更できません。finder メソッドのメソッド名（<method-name>）のテーブルの検索条件（<where-clause>）を設定してください。

Enterprise Bean 内の finder メソッド情報（<finder-impl>）は，次の項目で構成されています。

項目	対応するタグ名
finder メソッドのメソッド名	<method-name>
テーブルの検索条件	<where-clause>

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」を参照してください。

(4)　注意事項

- Entity Bean の CMP フィールドをデータベースの複数の表にマッピングする機能は提供していません。

- マッピングで利用する DB Connector の定義で指定されているユーザと，CMP のマッピング先となるデータベースの表の所有者が異なる場合，DB Connector の定義で指定されているユーザには次に示す権限が必要です。

  マッピング時：SELECT

  実行時：SELECT，INSERT，UPDATE，DELETE

9.6.3 CMP2.x とデータベースのマッピング

CMP2.x Entity Bean のフィールドをデータベース上のテーブルにマッピングします。

また，ここでは，CMP を定義した二つの CMP2.x Entity Bean の間に関連（CMR）を定義する方法についても説明します。

CMP2.x とデータベースのマッピング手順を，次に示します。

1. CMP2.x Entity Bean 間に CMR の関係がある場合，CMP2.x Entity Bean 間に CMR を定義します。
2. CMP2.x Entity Bean のフィールドをデータベース上のテーブルにマッピングします。
3. CMP2.x Entity Bean に対して cjgencmpsql コマンドを実行して SQL 文を生成します。

(1) CMR の定義

CMP を定義した二つの CMP2.x Entity Bean の間に CMR の関係がある場合，CMP2.x Entity Bean 間に CMR を定義します。

(a) 編集する属性ファイル

EJB-JAR 属性ファイル

(b) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して EJB-JAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名> -c <EJB-JAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder -c C:¥home¥adder_ejb.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，EJB-JAR 属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名> -c <EJB-JAR属性ファイルパス>
```

実行例

```
cjsetappprop MyServer -name adder -type ejb -resname adder -c C:¥home¥adder_ejb.xml
```

注意事項

開始状態のアプリケーションに含まれる EJB-JAR 属性ファイルの取得はできますが，定義された関連情報の反映はできません。

(c) 編集する属性設定項目

二つの CMP2.x Entity Bean 間の関係を定義する（<relationships> - <ejb-relation>）プロパティ設定項目を次に示します。

項目	必須	対応するタグ名
説明	△	<description>
関連定義名	○	<ejb-relation-name>
Enterprise Bean（EJB1）情報	○	<ejb1>
Enterprise Bean（EJB2）情報	○	<ejb2>

（凡例） ○：必須 △：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.3.1 EJB-JAR 属性ファイルの指定内容」を参照してください。

Enterprise Bean（EJB1）情報（<ejb1>）と Enterprise Bean（EJB2）情報（<ejb2>）について，それぞれ，設定側の Enterprise Bean と相手側の Enterprise Bean の関連についての項目を指定します。

項目	必須	対応するタグ名
説明	△	<description>
関連のロール名	△	<ejb-relationship-role-name>
設定側の多重度	○	<multiplicity>
カスケードデリートの設定	○	<cascade-delete>
設定側の<ejb-name>	○	<ejb-name>
CMR フィールドの名称	○	<cmr-field-name>
CMR フィールドの型	△	<cmr-field-type>

（凡例） ○：必須 △：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.3.1　EJB-JAR 属性ファイルの指定内容」を参照してください。

(2)　CMP2.x とデータベースのマッピング

CMP2.x Entity Bean のフィールドをデータベース上のテーブルにマッピングします。

(a)　編集する属性ファイル

編集する属性ファイルについては，「9.6.2　CMP1.x とデータベースのマッピング」の「(1)　編集する属性ファイル」を参照してください。

(b)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの所得と属性の設定については，「9.6.2　CMP1.x とデータベースのマッピング」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(c)　編集する属性設定項目

CMP2.x Entity Bean のフィールドをデータベース上の表にマッピングする（<cmp-map>）プロパティ設定項目を次に示します。

項目	必須	対応するタグ名
リソースアダプタの表示名	○	<datasource-name>
データベースのカタログ名	△	<catalog-name>
データベースのスキーマ名	△	<schema-name>
データベースのテーブル名	○	<table-name>
データベースへの書き込み許可／禁止	○	<read-only-access>
トランザクション遮断レベル※1	△	<transaction-isolation>
データベース書き込みデータ照合方法	△	<concurrency-protection>
フィールドとテーブルのカラムとのマッピング情報※2	○	<field-impl>

（凡例）　○：必須　△：任意

注※1

使用できるトランザクション遮断レベル（<transaction-isolation>）の値は，データベースおよび JDBC ドライバでサポートされるオプションによって異なります。

注※2

プライマリキーに対してデータベーステーブルの列を設定します。同じデータベーステーブルのフィールドのマッピングにも使用されます。

EntityBean のフィールド名（<field-name>）の値は変更できません。EntityBean のフィールド名（<field-name>）のマッピング先のデータベースの列（<column-name>）を設定してください。

フィールドとテーブルのカラムとのマッピング情報（<field-impl>）は，次の項目で構成されています。

項目	対応するタグ名
EntityBean のフィールド名	<field-name>
テーブルのカラム名	<column-name>

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」を参照してください。

(d)　注意事項

- Entity Bean の CMP フィールドをデータベースの複数の表にマッピングする機能は提供していません。
- マッピングで利用する DB Connector の定義で指定されているユーザと，CMP のマッピング先となるデータベースの表の所有者が異なる場合，DB Connector の定義で指定されているユーザには次に示す権限が必要です。

  マッピング時：SELECT

  実行時：SELECT，INSERT，UPDATE，DELETE
- CMP フィールドのマッピングが終わっていない場合，およびアプリケーションが開始状態の場合は，SQL を生成できません。
- CMR の関係を結ぶ二つの Bean のテーブルは同じデータベース製品のデータベース上になくてはいけません。

(3)　SQL 文の生成

finder/select メソッドが実行されたときに使用される SQL 文を生成します。

CMP2.x Entity Bean 間に CMR の関係がない場合

定義された EJB QL（<query>）を基に SQL 文を生成します。

CMP2.x Entity Bean 間に CMR の関係がある場合

SQL 文の生成をする前に，まず，すべての CMP2.x Entity Bean についてフィールドを表の列にマッピングします。すべての Bean についてマッピング操作の終了後，すべての CMP2.x Entity Bean について SQL 文を生成してください。Entity Bean に finder/select メソッドがない場合でも同様に SQL 文を生成してください。

なお，Entity Bean 間に CMR の関係がある場合は，finder/select メソッドが実行されたときに使用される SQL 文のほかに，CMR 用の表を操作するための SQL 文も生成されます。一度 SQL を生成したあとに CMR の設定を変更した場合は，開始する前に，変更した CMR に関する Entity Bean について SQL 文を生成してください。CMR 用の表の詳細については，「(4)　CMR 用の表」を参照してください。

次に示すコマンドを実行して SQL 文を生成します。

- J2EE アプリケーションに含まれるすべての CMP2.x Entity Bean の SQL 文を生成する場合

実行形式

```
cjgencmpsql ［<サーバ名称>］ -name J2EEアプリケーション名
```

実行例

```
cjgencmpsql MyServer -name App1
```

- J2EE アプリケーションに含まれる，特定の CMP2.x Entity Bean の SQL 文を生成する場合

実行形式

```
cjgencmpsql ［<サーバ名称>］ -name J2EEアプリケーション名 -resname <EJB-JARの表示名/Entity Beanの表示名>
```

実行例

```
cjgencmpsql MyServer -name App1 -resname EjbJar1/Ejb1
```

cjgencmpsql コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgencmpsql（CMP2.x Entity Bean 用 SQL 文の生成)」を参照してください。

(4) CMR 用の表

ここでは，CMR 用の表について説明します。

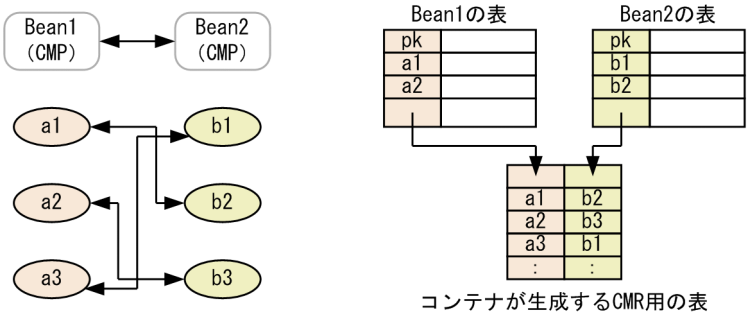
なお，以降の説明では，EJB-JAR の<ejb1> - <ejb-name>で設定されている Entity Bean を「Bean1」，<ejb2> - <ejb-name>で設定されている Entity Bean を「Bean2」と呼びます。

(a) CMR 用の表の概要

EJB コンテナでは，CMP2.x Entity Bean の間に CMR の関係があった場合，データベース上に表を作り，CMR の状態の保持のために利用します。

次の図に示すように CMR 用の表には関係を結ぶ二つの Entity Bean のプライマリキーが格納されます。

図 9-1 CMR 用の表（1 対 1，双方向の関係の場合）



(b) CMR 用の表の生成と削除

CMR を含むアプリケーションを開始すると，CMR 用の表が生成されます。生成される場所は Bean1 の表があるスキーマです。この表はアプリケーションの停止時に削除されます。表の生成，削除，操作のための SQL は，アプリケーションをカスタマイズするときのデータベースマッピングのあとで生成された SQL を使用します。

アプリケーションが開始状態のままサーバを停止させた場合は CMR 用の表はデータベース上に残ります。また，アプリケーションが開始状態のままサーバが再起動された場合は，CMR 用の表がデータベース上にあるかどうかをチェックします。アプリケーションの開始時およびサーバ起動時の CMR 用の表生成処理を次の表に示します。

表 9-4 アプリケーションの開始時およびサーバ起動時の CMR 用の表生成処理

状態	作成する表と同名の表がデータベース上にあるかどうか	同名の表がある場合，列の数，名前，JDBC 型が同じかどうか	動作	出力メッセージ ID
開始時※1	なし	該当しない	表を生成し，利用します。	KDJE43007-I
	あり	同じまたは同じでない	開始を中断します。※2	KDJE43003-E
J2EE サーバ起動時	なし	該当しない	開始を中断します。※3	KDJE43008-E
	あり	同じ	開始時に CMR 用として作ったものであると判断し，利用します。	KDJE43006-I
		同じでない	開始を中断します。※3	KDJE43008-E

注※1

CMR を含むアプリケーションの開始時のオプションを特に設定していないデフォルトの場合です。開始時のオプションについては，「(e) 障害発生時の CMR 用の表の回復」を参照してください。

注※2

同じ名前の表がすでにデータベース上にあるとアプリケーションの開始に失敗します。すでにある表が不要の場合，データベース製品で提供しているツールを使用して手動で削除するか，SQL 生成を再度実行してください。SQL 生成を再度実行すると，データベーススキーマ上の表と重複しない名前が表に付けられ，SQL が生成し直されます（SQL 生成は，EJB-JAR 内のすべての EJB のすべてのメソッドについて行ってください。これを行わないと表名の変更がすべての SQL に反映されないおそれがあります）。そのあと，再度アプリケーションを開始してください。

注※3

対象アプリケーションが開始されていない状態で J2EE サーバが起動します。

(c) CMR 用の表の命名規則

CMR 用の表の名前は次のようになります。

- 双方向の関係の場合

  [Bean1 の<ejb-name>] _ [Bean1 の<cmr-field-name>] _ [Bean2 の<ejb-name>] _ [Bean2 の<cmr-field-name>]

- 単方向の関係の場合（方向は Bean1 から Bean2 とします）

［Bean1 の<ejb-name>］_［Bean1 の<cmr-field-name>］_［Bean2 の<ejb-name>］

ただし，この方法で決めた名前が 29 文字以上になった場合は，28 文字になるように切ります。28 文字になるように切った名前がデータベース上の既存の表の名前や同じ EJB-JAR 内のほかの CMR 用の表の名前と重なる場合は，0～99 の番号を末尾に付けて区別します。

(d) CMR についての注意事項

- ユーザは Bean1 の表があるスキーマで，CREATE TABLE を行う権限が必要です。
- データベースや OS の種別によってはプライマリキーのサイズの制限で，CMR 用の表の生成に失敗することがあります。図 9-1 に示すように，CMR 用の表には Entity Bean のプライマリキーを格納するので，Entity Bean のプライマリキーのサイズを調整してください。
- アプリケーションの停止時に CMR 用の表は削除されるため，それまでに結んだ relation の情報が失われてしまいます。また，CMR 用の表を手動で削除した場合も relation の情報は失われます。
- CMR 用の表の名前を付ける場合に 0～99 の 100 種類の番号を使い切ったときは，CMR 用の表が生成できません。データベース上の不要な表を削除してその名前を使用できるようにするか，または「(c) CMR 用の表の命名規則」を参考にして異なる名前を付けられるようにしてください。
- relation を結ぶ Entity Bean のプライマリキーはデータベースの表のプライマリキーにできる型へマッピングしてください。
- CMR を含むアプリケーションの停止時に CMR 用の表は削除されますが，データベースへアクセスできないような障害発生などで CMR 用の表の削除に失敗する場合があります。この場合は，表がデータベース上に残るため，不要なときはデータベース製品で提供しているツールを使用して手動で削除してください。

(e) 障害発生時の CMR 用の表の回復

CMR を含むアプリケーションが開始され，運用していた状態で保守などのために J2EE サーバを停止，起動したときに障害が発生すると，CMR を含むアプリケーションが開始された状態で立ち上がらないことが考えられます。障害を解決し，再度 CMR を含むアプリケーションを開始しようとしても，表 9-4 に示されるように作成する表と同名の表がデータベース上にある場合，開始ができません（アプリケーション間で表の共有を避けるため）。

ここで，SQL 生成を再度実行すると，新しい CMR 用の表名を使用した SQL が生成され，新しい CMR 用の表を使用するように開始ができます。しかし，J2EE サーバを停止する以前に使用していた関係を継続させたい場合，データベースに残っている表を使うように開始する必要があります。

usrconf.properties ファイルの ejbserver.ejb.cmp20.cmr.use.existing_table キーは，アプリケーションの起動障害発生時，それまで使用していた関係の情報を回復させるためのオプションです。このキーに true を指定すると，データベース既存の表を使用するように開始できます。このキーに何も指定しない，または false を指定した場合は，表 9-4 の動作をします。このオプションを使用して，既存の表を使用する場合の手順を次に示します。

1. 障害発生前に使用していた CMR 用の表がデータベース上に残っていることを，データベース製品で提供しているツールなどを使用して確認してください。

2. J2EE サーバを停止します（アプリケーションが開始状態で立ち上がることに失敗した原因の対処をしてください）。

3. usrconf.properties ファイルに ejbserver.ejb.cmp20.cmr.use.existing_table=true を設定します。

4. J2EE サーバを起動します。

5. 開始に失敗した CMR を含むアプリケーションを再度開始します。

   注意事項

   ここで SQL 生成を再実行しないでください。再実行すると新しいテーブル名で SQL が生成され，以前の表が使用できなくなります。

6. J2EE サーバを停止します。

7. usrconf.properties ファイルに ejbserver.ejb.cmp20.cmr.use.existing_table=false を設定するか，またはこのオプションを設定しない状態に戻します。

8. 再度 J2EE サーバを起動します。

9.7　サーブレットと JSP のリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）のリファレンス（リソース参照）を定義します。次に示すリファレンスがあります。

- Enterprise Bean リファレンス
  Enterprise Bean 呼び出しのための参照です。
- リソースリファレンス
  データベースまたはメールサーバへの参照です。メールリファレンス，リソースアダプタリファレンス，およびリソース環境リファレンスがあります。

9.7.1　Enterprise Bean のリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）が Enterprise Bean を呼び出している場合，リファレンスを解決するためのプロパティを設定します。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得
  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder -c C:¥home¥adder_war.xml
  ```

- 属性の設定
  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder -c C:¥home¥adder_war.xml
  ```

(3)　編集する属性設定項目

Enterprise Bean が，ほかの Enterprise Bean を呼び出している場合のプロパティ設定項目と同じです。「9.3.1　ほかの Enterprise Bean のリファレンス定義」の「(3)　編集する属性設定項目」を参照してください。

9.7.2　メールコンフィグレーションのリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）が，メールコンフィグレーションを参照している場合，リファレンスを定義するためのプロパティを設定します。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの所得と属性の設定については，「9.7.1　Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

Enterprise Bean がメールコンフィグレーションを参照している場合のプロパティ設定項目と同じです。「9.3.2　メールコンフィグレーションのリファレンス定義」の「(3)　編集する属性設定項目」を参照してください。

9.7.3　リソースアダプタのリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）が，リソースアダプタを参照している場合，リファレンスを定義するためのプロパティを設定します。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの所得と属性の設定については，「9.7.1　Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

Enterprise Bean がリソースアダプタを参照している場合のプロパティ設定項目と同じです。「9.3.3　リソースアダプタのリファレンス定義」の「(3)　編集する属性設定項目」を参照してください。

9.7.4　リソース環境のリファレンス定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）がリソース環境を参照している場合，リファレンスを定義するためのプロパティを設定します。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

編集する属性ファイルの所得と属性の設定については，「9.7.1　Enterprise Bean のリファレンス定義」の「(2)　編集する属性ファイルの取得と属性の設定」を参照してください。

(3)　編集する属性設定項目

Enterprise Bean がリソース環境を参照している場合のプロパティ設定項目と同じです。「9.3.4　リソース環境のリファレンス定義」の「(3)　編集する属性設定項目」を参照してください。

9.8 サーブレットと JSP のマッピング定義

9.8.1 定義方法

サーブレットおよびJSP のマッピングを定義します。

(1) 編集する属性ファイル

サーブレット属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行してサーブレット属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名>/<サーブレットおよびJSPの表示名> -c <サーブレット属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder/adder_sv -c C:¥home¥adder_war.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，サーブレット属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名>/<サーブレットおよびJSPの表示名> -c <サーブレット属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder/adder_sv -c C:¥home¥adder_war.xml
  ```

(3) 編集する属性設定項目

項目	必須	対応するタグ名
URL パターン	○	<url-pattern>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.9.1　サーブレット属性ファイルの指定内容」を参照してください。

9.9　フィルタの設定

フィルタの設定方法について説明します。フィルタを設定するには，フィルタを WAR ファイルに追加して，マッピングを定義します。

9.9.1　フィルタの追加

次の手順で，WAR ファイルにフィルタを追加します。

1. フィルタ属性ファイルを作成します。

   テキストエディタなどを使用して，フィルタ属性ファイルを作成します。フィルタ属性ファイルには，フィルタ名称およびフィルタのクラス名を指定します。また，必要に応じて，初期化パラメタの名称および初期化パラメタの値を指定します。

   フィルタ属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.8　フィルタ属性ファイル」を参照してください。

2. 次に示すコマンドを実行して J2EE アプリケーション内の WAR ファイルにフィルタを登録します。

   実行形式

   ```
   cjaddapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -type filter -name ＜J2EEアプリケーション名＞ -warname ＜フィルタを追加するWARの表示名＞ -c ＜フィルタ属性ファイルパス＞
   ```

   実行例

   ```
   cjaddapp MyServer -type filter -name App1 -warname account-war -c FilterProp.xml
   ```

   cjaddapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjaddapp（リソースの追加)」を参照してください。

9.9.2　フィルタのマッピング定義

フィルタの追加後，フィルタのマッピングを定義します。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WARの表示名> -c <WAR属性ファイルパス>
```

実行例

```
cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
```

- 属性の設定

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WARの表示名> -c <WAR属性ファイルパス>
```

  実行例

```
cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
```

(3)　編集する属性設定項目

Web アプリケーション（サーブレットおよび JSP）のフィルタマッピング定義（<filter-mapping>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
フィルタ名	○	<filter-name>
URL パターン	○※2	<url-pattern>
サーブレット名	○※2	<servlet-name>
フィルタの適用条件※1	△	<dispatcher>

（凡例）　○：必須　△：任意

注※1　Servlet2.3 以前の WAR には，この属性は設定できません。

注※2　フィルタのマッピングは，URL パターンまたはサーブレット名のどちらかを指定してください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1　WAR 属性ファイルの指定内容」を参照してください。

9.9.3　フィルタの削除

次に示すコマンドを実行してフィルタを削除します。

実行形式

```
cjdeleteapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type filter -resname ＜削除するWARの表示名＞/＜削除するフィルタの表示名＞
```

WAR ファイルごとフィルタを削除する場合は，削除する WAR の表示名を指定します。フィルタだけを削除する場合は，削除する WAR の表示名とフィルタの表示名を指定します。

実行例

```
cjdeleteapp MyServer -name App1 -type filter -resname account-war/account-filter
```

cjdeleteapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeleteapp（J2EE アプリケーションの削除)」を参照してください。

9.10　Enterprise Bean の実行時属性の定義

次に示す Enterprise Bean の実行時属性を定義します。

- Stateful Session Bean
- Stateless Session Bean
- Entity Bean
- Message-driven Bean

9.10.1　Stateful Session Bean の実行時プロパティの設定

アプリケーションを構成する個々の Stateful Session Bean に対して，アプリケーション実行時のプロパティを設定します。

(1)　編集する属性ファイル

Session Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Session Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateful Session Bean表示名> -c <Session Bean属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder/MyAdder -c C:¥home¥MyAdder.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Session Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateful Session Bean表示名> -c <Session Bean属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type ejb -resname adder/MyAdder -c C:¥home¥MyAdder.xml
  ```

(3)　編集する属性設定項目

Stateful Session Bean の実行時プロパティは，次の二つに分けられます。

- Session Bean 共通の実行時属性
- Stateful Session Bean 固有の属性

ぞれぞれの設定項目を次に示します。

(a)　Session Bean 共通の実行時属性

Session Bean 共通の実行時属性（<runtime>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
Enterprise Bean のルックアップ名※1	○	<lookup-name>
リモートインタフェースを持つ Enterprise Bean 名の別名※1	△	<optional-name>
ローカルインタフェースを持つ Enterprise Bean 名の別名※1	△	<local-optional-name>
セッション最大数※2	△	<maximum-sessions>
参照渡しの設定※3	△	<pass-by-reference>

（凡例）　○：必須　△：任意

注※1

JNDI 名前空間に登録される名称の参照と変更については，「9.13.2　Enterprise Bean 名の参照と変更」を参照してください。

注※2

Stateful Session Bean に指定された最大セッション数は，EJB クライアントアプリケーションから利用できる Stateful Session Bean の最大数です。「0」を指定した場合，同時に生成できる Stateful Session Bean のセッション（EJB オブジェクト）の数は無制限（実際の最大値は 2147483647）になります。

注※3

参照渡しの設定は，usrconf.properties ファイルでも実施できます。プロパティまたは usrconf.properties ファイルのどちらかで設定されていれば，参照渡しの設定は有効になります。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

(b)　Stateful Session Bean 固有の属性

Stateful Session Bean 固有の実行時属性（<runtime> - <stateful>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
アクティブセッションの最大数	○	<maximum-active-sessions>
再び活性化するまでに，非活性化状態に保持している時間	○	<inactivity-timeout>
セッションが削除されるまでに非活性化状態に保持している時間	○	<removal-timeout>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

プロパティ項目の設定と動作について，次に説明します。

アクティブセッションの最大数

生成された Stateful Session Bean の状態には，次の二つの状態があります。

- method-ready 状態
- passive 状態

method-ready 状態の Stateful Session Bean は，EJB クライアントアプリケーションからアクセスされた時点ですでに実行できるようになっています。

passive 状態の Stateful Session Bean は，活性化され method-ready 状態になってから実行できます。

アクティブセッションの最大数に，同時に method-ready 状態になることができる Stateful Session Bean の最大数を指定します。この値は，複数の EJB クライアントアプリケーションからアクセスされたときに，すぐに実行開始できる Stateful Session Bean の最大数です。ここで指定した数以上の Stateful Session Bean が生成された場合，method-ready 状態の Stateful Session Bean が幾つか非活性化され passive 状態になります。ただし，このとき，method-ready 状態の Stateful Session Bean の中でトランザクション中のものについては，非活性化の対象になりません。セッション最大数，またはアクティブセッションの最大数に「0」を指定した場合は，非活性化の機能は動作しません（Stateful Session Bean が method-ready 状態から passive 状態に移りません）。

再び活性化するまでに，非活性化状態に保持している時間（デフォルト：0（分））

タイムアウト値を分単位で指定します。ここで指定された時間が経過するまでに passive 状態の Stateful Session Bean が活性化されない場合，この Bean は削除されます。再び活性化するまでに，非活性化状態に保持している時間に「0」を設定した場合は，タイムアウトは発生しません。また，Stateful Session Bean のインスタンスの削除は，指定した<inactivity-timeout>値に対し，最大で ejbserver.container.passivate.scan.interval に指定した時間分の遅延が生じることがあります。この起動間隔を変更したい場合は，usrconf.properties ファイルのシステムプロパティ ejbserver.container.passivate.scan.interval に起動間隔を秒単位で指定してください。デフォルトでは 0 秒が設定されています。ejbserver.container.passivate.scan.interval の指定例を次に示します。

10 分間隔で passive 状態の Stateful Session Bean の削除機能を起動したい場合

```
ejbserver.container.passivate.scan.interval=600
```

セッションが削除されるまでに非活性化状態に保持している時間（デフォルト：0（分））

タイムアウト値を分単位で指定します。method-ready 状態の Stateful Session Bean がここで指定された時間の間，一度も使用されない場合，この Bean は削除されます。ただし，このとき，method-ready 状態の Stateful Session Bean の中でトランザクション中のものについては，削除の対象になりません。

セッションが削除されるまでに非活性化状態に保持している時間に「0」を設定した場合は，タイムアウトは発生しません。また，Stateful Session Bean のインスタンスの削除は，指定した<removal-

timeout>値に対し，最大で ejbserver.container.remove.scan.interval に指定した時間分の遅延が生じることがあります。この起動間隔を変更したい場合は，J2EE サーバ用の usrconf.properties ファイルの ejbserver.container.remove.scan.interval に起動間隔を分位で指定してください。デフォルト値には 5 分が仮定されています。ejbserver.container.remove.scan.interval の指定例を次に示します。

10 分間隔で method-ready 状態の Stateful Session Bean の削除機能を起動したい場合

```
ejbserver.container.remove.scan.interval=10
```

(4) 注意事項

- デフォルトのユーザ定義では，「ejbActivate，ejbPassivate を呼び出し，Stateful Session Bean の状態を保存，復元する機能」であり，デフォルトでは，Stateful Session Bean の活性化，非活性化は動作しません。したがって，Stateful Session Bean の Enterprise Bean 実行時プロパティとして設定できる，<maximum-active-sessions>の設定は有効になりません。また，非活性化された Stateful Session Bean をタイムアウトで削除する機能は，動作しません。したがって，Stateful Session Bean の Enterprise Bean 実行時プロパティとして設定できる，<inactivity-timeout>の設定は有効になりません。
- 活性化，非活性化の機能を利用する場合は，J2EE サーバ用の usrconf.properties ファイルで，ejbserver.stateful.passivate.switch キーに対し「true」を指定します。
- 非活性化が動作するとき，Stateful Session Bean の状態はファイルに書き込むことで保存され，インスタンスは破棄されます。逆に活性化が動作するときはファイルから Stateful Session Bean の状態が読み込まれインスタンスが復元されます。活性化，非活性化が動作する場合，Stateful Session Bean（またはそれが参照するクラス）のメンバ変数に設定されているオブジェクトも保存，復元の対象になります。このとき，保存，復元ができるものを次に示します。
  - 直列化できるオブジェクト
  - EJB オブジェクトリファレンス
  - EJB ホームオブジェクトリファレンス
  - "java:comp/env"のルックアップで取得した javax.naming.Context
  - javax.ejb.SessionContext

  なお，javax.jta.UserTransaction は保存，復元できないため，メンバ変数に保持しないでください。保存，復元ができないオブジェクトをメンバ変数に保持している場合は，ejbPassivate で解放し（メンバ変数に null 代入），ejbActivate で再取得してください。
- 同時に生成できる Stateful Session Bean のセッションの最大数と method-ready 状態の Stateful Session Bean の最大数は次の条件を満たすように指定してください。

  ```
  method-ready状態のStateful Session Beanの最大数≦同時に生成できるStateful Session Beanのセッションの最大数
  ```

  また，同時に生成できる Stateful Session Bean のセッションの最大数を指定した場合，method-ready 状態の Stateful Session Bean の最大数も指定する必要があります。

9.10.2　Stateless Session Bean の実行時プロパティの設定

アプリケーションを構成する個々の Stateless Session Bean に対して，アプリケーション実行時のプロパティを設定します。

(1)　編集する属性ファイル

Session Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Session Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateless Session Bean表示名> -c <Session Bean属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder/MyAdder -c C:¥home¥MyAdder.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Session Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateless Session Bean表示名> -c <Session Bean属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type ejb -resname adder/MyAdder -c C:¥home¥MyAdder.xml
  ```

(3)　編集する属性設定項目

Stateless Session Bean の実行時プロパティは，次の三つに分けられます。

- Session Bean 共通の実行時属性
- Stateless Session Bean 固有の属性
- CTM 連携の実行時属性

ぞれぞれの設定項目を次に示します。

(a) Session Bean 共通の実行時属性

Session Bean 共通の実行時属性（<runtime>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
Enterprise Bean のルックアップ名※1	○	<lookup-name>
リモートインタフェースを持つ Enterprise Bean 名の別名※1	△	<optional-name>
ローカルインタフェースを持つ Enterprise Bean 名の別名※1	△	<local-optional-name>
セッション最大数※2	△	<maximum-sessions>
スケジューリングの設定※3	△	<enable-scheduling>
参照渡しの設定※4	△	<pass-by-reference>

（凡例） ○：必須　△：任意

注※1

JNDI 名前空間に登録される名称の参照と変更については，「9.13.2　Enterprise Bean 名の参照と変更」を参照してください。

注※2

Stateless Session Bean に指定された最大セッション数は，クライアントが使用できる Stateless Session Bean インスタンス数を定義します。ただし，この指定値は有効にはなりません。

注※3

CTM のスケジューリングについては，「9.14　CTM のスケジューリング」を参照してください。

注※4

参照渡しの設定は，usrconf.properties ファイルでも実施できます。プロパティまたは usrconf.properties ファイルのどちらかで設定されていれば，参照渡しの設定は有効になります。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

(b) Stateless Session Bean 固有の属性

Stateless Session Bean 固有の実行時属性（<runtime> - <stateless>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
プール内のインスタンスの最小値	○	<pooled-instance> - <minimum>
プール内のインスタンスの最大値	○	<pooled-instance> - <maximum>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

プロパティ項目の設定と動作について，次に説明します。

プール内のインスタンスの最小値

J2EE アプリケーション起動時にここで指定した数の Stateless Session Bean が生成され，プーリングされます。プーリングされる Stateless Session Bean の数は，EJB クライアントアプリケーションからのアクセス量に応じて，最大数と最小数の間になります。プール内のインスタンスの最小値に「0」を指定した場合は，J2EE アプリケーション起動時に Stateless Session Bean が生成されません。

この値は，アプリケーションが開始されたときに作成する Stateless Session Bean インスタンスの数も指定します。

プール内のインスタンスの最大値

生成された Stateless Session Bean は，method-ready でプーリングされます。プーリングされた Stateless Session Bean は，EJB クライアントアプリケーションからアクセスされた時点ですでに実行できるようになっています。プール内のインスタンスの最大値に，プーリングされる Stateless Session Bean の最大数を指定します。この値は，複数の EJB クライアントアプリケーションからアクセスされたときに，すぐに実行開始できる Stateless Session Bean の最大数です。プール内のインスタンスの最大値に 0 を指定した場合は，プーリングされる Stateless Session Bean の数に制限はありません。

(c)　CTM 連携の実行時属性

CTM 連携の実行時属性（<runtime> - <scheduling>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
キュー名	○	<queue-name>
スレッド数	○	<parallel-count>

（凡例）　○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

CTM のスケジューリングについては，「9.14　CTM のスケジューリング」を参照してください。

(4)　注意事項

- クライアント側からアクセスできる Stateless Session Bean の上限数をチェックする機能は動作しません。代わりに＜プール内のインスタンスの最大値＞で設定します。
- プーリングされる Stateless Session Bean の最小数とプーリングされる Stateless Session Bean の最大数は，次の条件を満たすように指定してください。

  プーリングされるStateless Session Beanの最小数≦プーリングされるStateless Session Beanの最大数

  また，プーリングされる Stateless Session Bean の最大数を指定した場合，プーリングされる Stateless Session Bean の最小数も指定する必要があります。

9.10.3　Entity Bean の実行時プロパティの設定

アプリケーションを構成する個々の Entity Bean に対して，アプリケーション実行時のプロパティを設定します。

(1)　編集する属性ファイル

Entity Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Entity Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名> -c <Entity Bean属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name account -type ejb -resname account/MyAccount -c C:¥home¥MyAccount.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，Entity Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Entity Bean表示名> -c <Entity 属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name account -type ejb -resname account/MyAccount -c C:¥home¥MyAccount.xml
  ```

(3)　編集する属性設定項目

Entity Bean の実行時属性（<runtime>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
Enterprise Bean のルックアップ名※1	○	<lookup-name>
リモートインタフェースを持つ Enterprise Bean 名の別名※1	△	<optional-name>
ローカルインタフェースを持つ Enterprise Bean 名の別名※1	△	<local-optional-name>
セッション最大数※2	△	<maximum-instances>

項目	必須	対応するタグ名
参照渡しの設定※3	△	<pass-by-reference>
プール内のインスタンスの最大値	○	<pooled-instance> - <maximum>
プール内のインスタンスの最小値	○	<pooled-instance> - <minimum>
データのキャッシュ方法	○	<caching-model>
EJB オブジェクトの存在期限	△	<entity-timeout>

（凡例） ○：必須　△：任意

注※1

JNDI 名前空間に登録される名称の参照と変更については，「9.13.2　Enterprise Bean 名の参照と変更」を参照してください。

注※2

Entity Bean に指定された最大セッション数は，EJB クライアントアプリケーションから利用できる Entity Bean の最大値です。0 を指定した場合は，同時に生成できる Entity Bean のリモートオブジェクトの数は無制限（実際の最大値は 2147483647）になります。

注※3

参照渡しの設定は，usrconf.properties ファイルでも実施できます。プロパティまたは usrconf.properties ファイルのどちらかで設定されていれば，参照渡しの設定は有効になります。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」を参照してください。

プロパティ項目の設定と動作について，次に説明します。

プール内のインスタンスの最大値

メモリ上でプーリングされる Entity Bean には，次の二つの状態があります。

- ready 状態

  ready 状態の Entity Bean は，データがデータベース上からインスタンス中に読み込まれた状態のもので，Entity Bean としてのアイデンティティを持っています。ready 状態のものは EJB クライアントアプリケーションからアクセスされた時点ですでに実行できるようになっています。

- pool 状態

  pool 状態の Entity Bean は，データがデータベース上からインスタンス中に読み込まれていない状態のもので，Entity Bean としてのアイデンティティを持っていません。pool 状態のものは，一度活性化され ready 状態になってから実行できます。

ready 状態の Entity Bean が多くなると，幾つかが非活性化され pool 状態になります。ただし，このとき，ready 状態の Entity Bean の中でトランザクション処理中のものについては，非活性化の対象になりません。

プール内のインスタンスの最大値（<pooled-instance> - <maximum>）には，プーリングされる ready 状態と pool 状態の Entity Bean の最大数を指定します。これは，メモリ上に展開される Entity Bean インスタンスの上限です。

プール内のインスタンスの最大値に「0」を指定した場合は，プーリングされる Entity Bean インスタンスの数は無制限になります。

プールされたインスタンスの数がこの値を超えると，J2EE サーバは，最も活性でないインスタンスを非活性化します。

プール内のインスタンスの最小値

プーリングされる ready 状態と pool 状態の Entity Bean の最小数を指定します。この値は，メモリ上に展開される Entity Bean インスタンスの下限です。J2EE アプリケーション起動時にここで指定した数の Entity Bean が生成され，pool 状態，または ready 状態になりプーリングされます。プーリングされる Entity Bean の数は，EJB クライアントアプリケーションからのアクセス量に応じて，最大数と最小数の間になります。

プール内のインスタンスの最小値（<pooled-instance> - <minimum>）に 0 を指定した場合は，J2EE アプリケーション起動時に Entity Bean が生成されません。クライアントに参照されない Entity Bean インスタンスを，最低幾つメモリに保持しておくかを指定します。この値は，プール内のインスタンスの最大値（<pooled-instance> - <maximum>）の値を超えてはいけません。

データのキャッシュ方法

Entity Bean のキャッシュモデル（コミットオプション）を指定します。

- Full caching（commit option A）

  トランザクション開始時にデータベースから Entity Bean インスタンスにデータが読み込まれないため，Entity Bean が前回のトランザクションコミット時と同じ状態のままでトランザクションが開始されます（例えば，前回のトランザクションコミット時からトランザクション開始時の間にほかの J2EE サーバが Entity Bean を更新した場合，Entity Bean の状態の一貫性が保たれません）。Full caching は参照系の Entity Bean 用のキャッシュモデルです。

- Caching（commit cache option B）

  トランザクション開始時にデータベースから Entity Bean インスタンスにデータが読み込まれるため，Entity Bean がデータベースの最新状態と同じ状態でトランザクションが開始されます。Caching は更新系の Entity Bean 用のキャッシュモデルです。

- No caching（commit cache option C）

  トランザクションコミット時に Entity Bean が非活性化されます。トランザクション開始時には，一度活性化され，データベースから Entity Bean インスタンスにデータが読み込まれます。このため，Entity Bean がデータベースの最新状態と同じ状態でトランザクションが開始されます。No caching は更新系の Entity Bean 用です。また，トランザクションコミット時に必ず非活性化されるため，多数の Entity Bean を利用する場合のキャッシュモデルです。

EJB オブジェクトの存在期限

タイムアウト値を秒単位で指定します。値は，0 または正の整数値で指定します。パッシベイトされた Entity Bean の EJB オブジェクトは，最小でこのタイムアウト値に指定した時間，最大でこのタイムアウト値に指定した時間+usrconf.properties ファイルの ejbserver.container.passivate.scan.interval に指定した値の時間だけ存在します。クライアントからタイムアウト値の時間を経過してもアクセスされない場合，該当する EJB オブジェクトは削除されます。

EJB オブジェクトの存在期限（<entity-timeout>）に「0」を指定した場合は，タイムアウトは発生しません。

(4) 注意事項

- 同時に生成できる Entity Bean の EJB オブジェクトの最大数に上限を指定した場合，プーリングされる Entity Bean の最大数にも上限を指定しなければいけません。
- 同時に生成できる Entity Bean の EJB オブジェクトの最大数，プーリングされる Entity Bean の最大数，プーリングされる Entity Bean の最小数の値は，次の条件を満たすように指定してください。

```
プーリングされるEntity Beanの最小数≦プーリングされるEntity Beanの最大数≦同時に生成できるEntity BeanのEJBオブジェクトの最大数
```

9.10.4 Message-driven Bean の実行時プロパティの設定

アプリケーションを構成する個々の Message-driven Bean に対して，アプリケーション実行時のプロパティを設定します。

(1) 編集する属性ファイル

Message-driven Bean 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して Message-driven Bean 属性ファイルを取得します。

  実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Message-driven Bean表示名> -c <Message-driven Bean属性ファイルパス>
```

  実行例

```
cjgetappprop MyServer -name message -type ejb -resname message/MyMessageBean -c C:¥home¥MyMessageBean.xml
```

- 属性の設定

  次に示すコマンドを実行して，Message-driven Bean 属性ファイルの値を反映します。

  実行形式

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Message-driven Bean表示名> -c <Message-driven Bean属性ファイルパス>
```

実行例

```
cjsetappprop MyServer -name message -type ejb -resname message/MyMessageBean -c C:¥home¥MyMessageBean.xml
```

(3) 編集する属性設定項目

Message-driven Bean の実行時属性（<runtime>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
プール内のインスタンスの最大値※	○	<pooled-instance> - <maximum>

（凡例） ○：必須

注※ Server Session プールの初期化時に指定値まで Bean を生成します。1～2147483647（int 型の最大値）を指定できます。デフォルト値は 1 です。指定数分の JMS セッションを生成するため，JMS プロバイダで生成できる JMS セッション数に合わせて定義します。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1 MessageDrivenBean 属性ファイルの指定内容」を参照してください。

9.11　サーブレットと JSP の実行時属性の定義

J2EE アプリケーションを構成する Web アプリケーション（サーブレットおよび JSP）の実行時属性を定義します。次に示す定義があります。

- J2EE アプリケーションのコンテキストルートの定義
- Web アプリケーション単位での同時実行スレッド数制御の定義
- URL グループ単位での同時実行スレッド数制御の定義
- URL グループ単位の実行待ちリクエストの監視の定義

それぞれの定義で設定するプロパティ項目を次に示します。

9.11.1　J2EE アプリケーションのコンテキストルート定義

J2EE アプリケーションのコンテキストルートを定義します。

コンテキストルートには，ルートコンテキストも定義できます。**ルートコンテキスト**とは，コンテキストルートが空文字""のコンテキストです。ルートコンテキストに welcome ファイルを作成すると，「http://www.xxx.com/」のようなドメイン名だけの URL から J2EE アプリケーションのトップページを表示できるようになります。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

- **属性の設定**

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
```

実行例

```
cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
```

(3) 編集する属性設定項目

J2EE アプリケーションのコンテキストルート定義（<runtime>）のプロパティ設定項目を，次に示します。

項目	必須	対応するタグ名
コンテキストルート	○	<context-root>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1 WAR 属性ファイルの指定内容」を参照してください。

(4) 注意事項

- コンテキストルートには，"ejb/"，または"web/"から始まる文字列を指定しないでください。
  複数の J2EE アプリケーションでコンテキストルートのパスの構成要素が包含関係（例："test"と"test/jsp"）にある場合で，包含する方のパスを含む URL（例："/test/jsp/test.jsp"）を指定してアクセスしたときは，包含する方のアプリケーション（例では，"test/jsp"をコンテキストルートに持つ方）が有効になり，包含される方のアプリケーション（例では，"test"をコンテキストルートに持つ方）にはアクセスできません。
- コンテキストルートは，作業ディレクトリ中のディレクトリ名として用いられます。作業ディレクトリのパス長がプラットフォームの上限に達しないようにコンテキストルートを指定してください。作業ディレクトリのパス長の見積もりについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」の「付録 C.1 J2EE サーバの作業ディレクトリ」を参照してください。
- ルートコンテキストを使用して開始する Web アプリケーションでは，URL が"ejb"または"web"で始まる構成にしないでください。
- コンソールおよびログファイルに出力されるメッセージでは，ルートコンテキストのコンテキストルートは，空文字「""」で表示されます。

9.11.2 Web アプリケーション単位での同時実行スレッド数制御の定義

Web コンテナで同時実行スレッド数を制御するための定義をします。

(1) 編集する属性ファイル

WAR 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

(3) 編集する属性設定項目

Web コンテナで同時実行スレッド数を制御するためのプロパティ設定項目（<thread-control>）を次に示します。

項目	必須	対応するタグ名
最大同時実行スレッド数	○	<thread-control-max-threads>
占有して使用するスレッド数	○	<thread-control-exclusive-threads>
Web アプリケーション単位の実行待ちキューサイズ	○	<thread-control-queue-size>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1　WAR 属性ファイルの指定内容」を参照してください。

9.11.3　URL グループ単位での同時実行スレッド数制御の定義

URL グループ単位での同時実行スレッド数を制御するための定義をします。

URL グループ単位での同時実行スレッド数制御を有効にするためは，Web アプリケーション単位での同時実行スレッド数制御の定義を設定してください。Web アプリケーション単位での同時実行スレッド数制御については，「9.11.2　Web アプリケーション単位での同時実行スレッド数制御の定義」を参照してください。

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

(3)　編集する属性設定項目

URL グループ単位で同時実行スレッド数を制御するためのプロパティ設定項目（<urlgroup-thread-control>）を次に示します。

項目	必須	対応するタグ名
定義名	○	<urlgroup-thread-control-name>
最大同時実行スレッド数	○	<urlgroup-thread-control-max-threads>

項目	必須	対応するタグ名
占有スレッド数	○	<urlgroup-thread-control-exclusive-threads>
URL グループ単位の実行待ちキューサイズ	○	<urlgroup-thread-control-queue-size>
URL パターン	○	<urlgroup-thread-control-mapping> - <url-pattern>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1　WAR 属性ファイルの指定内容」を参照してください。

(4) 注意事項

- URL グループ単位での同時実行スレッド数制御で実行されるスレッド数は，Web アプリケーション単位で実行されるスレッド数に含まれています。したがって，URL グループ単位で一つスレッドが実行されている場合，この Web アプリケーションの Web アプリケーション単位でも一つスレッドが実行されていることになります。
- URL グループ単位での同時実行スレッド数は，次の範囲で設定します。
  - 最大同時実行スレッド数

    最大同時実行スレッド数の指定範囲を次に示します。

    ```
    1　≦　URLグループ単位の最大同時実行スレッド数の総和　≦　Webアプリケーション単位の最大同時実行スレッド数
    ```

  - 占有スレッド数

    占有スレッド数の指定範囲を次に示します。

    条件 1 と条件 2 は，すべて満たしている必要があります。

    ```
    条件1：
    ・Webアプリケーション単位の最大同時実行スレッド数＝Webアプリケーション単位の占有スレッド数の場合
      0　≦　URLグループ単位の占有スレッド数　≦　URLグループ単位の最大同時実行スレッド数
    ・Webアプリケーション単位の最大同時実行スレッド数≠Webアプリケーション単位の占有スレッド数の場合
      0　≦　URLグループ単位の占有スレッド数　＜　URLグループ単位の最大同時実行スレッド数

    条件2：
      URLグループ単位の占有スレッド数の総和　≦　Webアプリケーション単位の占有スレッド数
    ```

9.11.4 URL グループ単位の実行待ちリクエスト数の監視の定義

URL グループ単位の実行待ちリクエスト数を監視するための定義をします。

(1) 編集する属性ファイル

WAR 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

(3) 編集する属性設定項目

URL グループ単位の実行待ちリクエスト数を監視するためのプロパティ設定項目（<urlgroup-thread-control> - <stats-monitor> - <waiting-request-count>）を次に示します。

項目	必須	対応するタグ名
しきい値イベント監視有無	○	<enabled>
しきい値イベントの出力上限しきい値※	○	<high-threshold>
しきい値イベントの出力下限しきい値※	○	<low-threshold>

（凡例） ○：必須

注※ 「しきい値イベントを出力する上限しきい値≧しきい値イベントを出力する下限しきい値」となるよう指定してください。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1　WAR 属性ファイルの指定内容」を参照してください。

9.12　デフォルトの文字エンコーディングの設定

9.12.1　設定方法

Web アプリケーション単位にデフォルトの文字エンコーディングを設定します。

次の文字エンコーディングについて，デフォルトのエンコーディングが設定できます。

- リクエストボディおよびクエリの文字エンコーディング
- レスポンスボディの文字エンコーディング
- JSP ファイルの文字エンコーディング

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

- 属性ファイルの設定

  次に示すコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type war -resname adder_war -c C:¥home¥adder_war.xml
  ```

(3)　編集する属性設定項目

デフォルトの文字エンコーディングのプロパティ設定項目を次に示します。

項目	必須	対応するタグ名
リクエストボディおよびクエリの文字エンコーディング	△	<http-request> - <encoding>
レスポンスボディの文字エンコーディング	△	<http-response> - <encoding>
JSP ファイルの文字エンコーディング	△	<jsp> - <page-encoding>

（凡例） △：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1　WAR 属性ファイルの指定内容」を参照してください。

9.13　JNDI 名前空間に登録される名称の参照と変更

JNDI 名前空間に登録される J2EE アプリケーション名や Enterprise Bean 名を変更します。Enterprise Bean 名には，別名も付けられます。

EJB ホームオブジェクトは，J2EE アプリケーション開始時に JNDI 名前空間（HITACHI_EJB/SERVERS/＜サーバ名称＞/EJB/＜ J2EE アプリケーション名＞/＜ Enterprise Bean 名＞）に割り当てられます。この名前は，EJB クライアントアプリケーションから参照する場合やネーミングの切り替え機能を使って参照する場合に使用します。

EJB ホームオブジェクトには，別名（Optional Name）が付けられます。別名を付けることで，EJB クライアントアプリケーションは JNDI 名前空間から任意の名前で EJB ホームオブジェクトを取得できるようになります。なお，この機能を**ユーザ指定名前空間機能**といいます。

なお，EJB ホームオブジェクトの JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.3　JNDI 名前空間へのオブジェクトのバインドとルックアップ」を参照してください。ユーザ指定名前空間機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「2.6　Enterprise Bean または J2EE リソースへの別名付与（ユーザ指定名前空間機能)」を参照してください。

9.13.1　J2EE アプリケーション名の参照

J2EE アプリケーション名の参照は，次の手順で実行します。

1. アプリケーション属性ファイルを取得します。

   次に示すコマンドを実行して，アプリケーション属性ファイルを取得します。

   実行形式

   ```
   cjgetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -c ＜アプリケーション属性ファイルパス＞
   ```

   実行例

   ```
   cjgetappprop MyServer -name account -c C:¥home¥accountAPP.xml
   ```

2. テキストエディタなどでアプリケーション属性ファイルを開きます。

   アプリケーション属性ファイルの J2EE アプリケーション名（<lookup-name>）を参照します。

cjgetappprop コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetappprop（アプリケーションの属性の取得)」を参照してください。属性ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.2　アプリケーション属性ファイル」を参照してください。

注意事項

JNDI 名前空間に登録される J2EE アプリケーション名は J2EE アプリケーション名を基に自動的に割り当てられます。変更はできません。

9.13.2 Enterprise Bean 名の参照と変更

次の Enterprise Bean の Enterprise Bean 名を参照および変更します。

- Session Bean
- Entity Bean

(1) 編集する属性ファイル

- Session Bean 属性ファイル
- Entity Bean 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次のコマンドを実行して，Enterprise Bean 属性ファイル（Session Bean 属性ファイルまたは Entity Bean 属性ファイル）を取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Bean属性ファイルパス※>
  ```

  注※　Enterprise Bean 属性ファイルパスには，Session Bean 属性ファイルまたは Entity Bean 属性ファイルのファイルパスを指定します。

  実行例

  ```
  cjgetappprop MyServer -name account -type ejb -resname account/MyAccount -c C:¥home¥MyAccount.xml
  ```

- **属性ファイルの設定**

  次のコマンドを実行して，Enterprise Bean 属性ファイル（Session Bean 属性ファイルまたは Entity Bean 属性ファイル）の値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Bean属性ファイルパス※>
  ```

注※　Enterprise Bean 属性ファイルパスには，Session Bean 属性ファイルまたは Entity Bean 属性ファイルのファイルパスを指定します。

実行例

```
cjsetappprop MyServer -name account -type ejb -resname account/MyAccount -c C:¥home¥My
Account.xml
```

(3)　編集する属性設定項目

Enterprise Bean 属性ファイル（Session Bean 属性ファイルまたは Entity Bean 属性ファイル）の Enterprise Bean 名を，実行時属性（<runtime>）で参照および変更します。

項目	必須	対応するタグ名
Enterprise Bean のルックアップ名	○	<lookup-name>
リモートインタフェースを持つ Enterprise Bean 名の別名	△	<optional-name>
ローカルインタフェースを持つ Enterprise Bean 名の別名	△	<local-optional-name>

（凡例）　○：必須　△：任意

プロパティの設定項目については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1　Entity Bean 属性ファイルの指定内容」

(4)　注意事項

- usrconf.properties ファイルの ejbserver.cui.optionalname.enabled キーに true が設定されていて，かつ EJB がリモートインタフェースを実装している場合にだけ指定できます。
- JNDI 名前空間に登録される Enterprise Bean 名の初期値は，Enterprise Bean の ejb-name が使われます。
- Message-driven Bean には JNDI 名前空間に登録される Enterprise Bean 名がありません。

9.14　CTM のスケジューリング

CTM を利用する場合の，スケジューリングについて設定します。CTM は，構成ソフトウェアに Component Transaction Monitor を含む製品だけで利用できます。利用できる製品については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「2.2.1　製品と構成ソフトウェアの対応」を参照してください。

次の手順で，CTM のスケジューリングを設定します。

- アプリケーション単位のスケジューリングを設定します。
- スケジューリングの対象にしたい Stateless Session Bean のスケジューリングを設定します。

9.14.1　J2EE アプリケーション単位のスケジューリング

J2EE アプリケーション単位の CTM との連携にかかわる設定をします。

アプリケーションの属性と Session Bean の属性の両方を設定する必要があるので，アプリケーション統合属性ファイルを使用して設定することを推奨します。

(1)　編集する属性ファイル

アプリケーション統合属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行してアプリケーション統合属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type all -c <アプリケーション統合属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name account -type all -c C:¥home¥account.xml
  ```

- 属性ファイルの設定

  次に示すコマンドを実行して，アプリケーション統合属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type all -c <アプリケーション統合属性ファイルパス>
  ```

実行例

```
cjsetappprop MyServer -name account -type all -c C:¥home¥account.xml
```

(3)　編集する属性設定項目

編集する属性設定項目を次に示します。

- アプリケーションの属性
- Session Bean の属性

ぞれぞれの設定項目を次に示します。

(a)　アプリケーションの属性

J2EE アプリケーション単位の CTM との連携にかかわる設定項目を次に示します。

項目	必須	対応するタグ名
CTM 連携有無	○	<managed-by-ctm>
キューの配置モデル	○	<scheduling-unit>
キュー名	○	<scheduling> - <queue-name>
スレッド数	○	<scheduling> - <parallel-count>
キューの長さ	△	<scheduling> - <queue-length>

（凡例）　○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.2.1　アプリケーション属性ファイルの指定内容」を参照してください。

(b)　Session Bean の属性

J2EE アプリケーション単位の CTM との連携にかかわる実行時属性設定項目（<session-runtime>）を次に示します。

項目	必須	対応するタグ名
スケジューリングの設定	○	<enable-scheduling>

（凡例）　○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

(4) 注意事項

アプリケーション統合属性ファイルを使用しないで，アプリケーション属性ファイルおよび Session Bean 属性ファイルを使用して設定する場合は，次の順に設定する必要があります。

1. 次に示すコマンドを実行して，Session Bean 属性ファイルの<enable-scheduling>タグの値を反映します。

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateless Session Bean表示名> -c <Session Bean属性ファイルパス>
```

2. 次に示すコマンドを実行して，アプリケーション属性ファイルの<managed-by-ctm>タグ，<scheduling-unit>タグ，および<scheduling>タグの下位のタグの値を反映します。

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
```

9.14.2 Stateless Session Bean 単位のスケジューリング

CTM によるスケジューリングの対象に設定した J2EE アプリケーションに含まれている Stateless Session Bean を CTM によるスケジューリングの対象にする場合，Stateless Session Bean 単位のスケジューリングの設定をします。

スケジューリングの対象となるのは，リモートホームインタフェースを実装している Stateless Session Bean だけです。

スケジューリング機能を利用する場合，Stateless Session Bean の実行時プロパティを設定します。Stateless Session Bean の実行時プロパティの設定については，「9.10.2 Stateless Session Bean の実行時プロパティの設定」を参照してください。

また，アプリケーションの属性で CTM に連携するための設定をします。

アプリケーションの属性と Session Bean の属性の両方を設定する必要があるので，アプリケーション統合属性ファイルを使用して設定することを推奨します。

(1) 編集する属性ファイル

アプリケーション統合属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次に示すコマンドを実行してアプリケーション統合属性ファイルを取得します。

実行形式

```
cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type all -c <アプリケーション統合属性ファイルパス>
```

実行例

```
cjgetappprop MyServer -name adder -type all -c C:¥home¥Adder.xml
```

- 属性の設定

次に示すコマンドを実行して，アプリケーション統合属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type all -c <アプリケーション統合属性ファイルパス>
```

実行例

```
cjsetappprop MyServer -name adder -type all -c C:¥home¥Adder.xml
```

(3) 編集する属性設定項目

編集する属性設定項目を次に示します。

- アプリケーションの属性
- Session Bean の属性

ぞれぞれの設定項目を次に示します。

(a) アプリケーションの属性

CTM に連携するための設定項目を次に示します。

項目	必須	対応するタグ名
CTM 連携有無	○	<managed-by-ctm>
キューの配置モデル	○	<scheduling-unit>

（凡例） ○：必須

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.2.1 アプリケーション属性ファイルの指定内容」を参照してください。

(b) Session Bean の属性

CTM に連携するための実行時属性設定項目（<session-runtime>）を次に示します。

項目	必須	対応するタグ名
スケジューリングの設定	○	<enable-scheduling>

項目	必須	対応するタグ名
キュー名	○	<scheduling> - <queue-name>
スレッド数	○	<scheduling> - <parallel-count>
キューの長さ	△	<scheduling> - <queue-length>

（凡例） ○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1　Session Bean 属性ファイルの指定内容」を参照してください。

(4)　注意事項

アプリケーション統合属性ファイルを使用しないで，アプリケーション属性ファイルおよび Session Bean 属性ファイルを使用して設定する場合は，次の順に設定する必要があります。

1. 次に示すコマンドを実行して，Session Bean 属性ファイルの<enable-scheduling>タグの値を反映します。

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateless Session Bean表示名> -c <Session Bean属性ファイルパス>
```

2. 次に示すコマンドを実行して，アプリケーション属性ファイルの<managed-by-ctm>タグおよび<scheduling-unit>タグの値を反映します。

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
```

3. 次に示すコマンドを実行して，Session Bean 属性ファイルの<scheduling>タグの下位のタグの値を反映します。

```
cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Stateless Session Bean表示名> -c <Session Bean属性ファイルパス>
```

9.15　起動順序の設定

J2EE アプリケーションの起動順序，および J2EE アプリケーションに含まれる Enterprise Bean および WAR の起動順序を設定します。

9.15.1　J2EE アプリケーションの起動順序の設定

J2EE アプリケーションの起動順序を設定します。

(1)　編集する属性ファイル

アプリケーション属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行してアプリケーション属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name account -c C:¥home¥MyAccount.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，アプリケーション属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -c <アプリケーション属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name account -c C:¥home¥MyAccount.xml
  ```

(3)　編集する属性設定項目

J2EE アプリケーションの起動順序設定項目を次に示します。

項目	必須	対応するタグ名
開始・停止順※	△	<start-order>

（凡例）　△：任意

注※　値が小さいほど，先に起動します。また，値が大きいほど，先に停止します。なお，複数の J2EE アプリケーションに対して同じ値を指定した場合，それらの間の起動順序は保証されません。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.2.1　アプリケーション属性ファイルの指定内容」を参照してください。

9.15.2　Enterprise Bean の起動順序の設定

Enterprise Bean の起動順序を設定します。

(1)　編集する属性ファイル

- Session Bean 属性ファイル
- Entity Bean 属性ファイル
- Message-driven Bean 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次のコマンドを実行して，編集する Enterprise Bean 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Bean属性ファイルパス※>
  ```

  注※　Enterprise Bean 属性ファイルパスには，編集する属性ファイルパスを指定します。

  実行例

  ```
  cjgetappprop MyServer -name account_eb -type ejb -resname account/account -c C:¥home¥MyAccount.xml
  ```

- **属性の設定**

  次のコマンドを実行して，編集した Enterprise Bean 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type ejb -resname <EJB-JAR表示名>/<Enterprise Bean表示名> -c <Enterprise Bean属性ファイルパス※>
  ```

  注※　Enterprise Bean 属性ファイルパスには，値を反映する属性ファイルパスを指定します。

  実行例

  ```
  cjsetappprop MyServer -name account -type ejb -resname account/account_eb -c C:¥home¥MyAccount.xml
  ```

(3) 編集する属性設定項目

Enterprise Bean の起動順序設定項目を次に示します。

項目	必須	対応するタグ名
開始・停止順※	△	<start-order>

（凡例） △：任意

注※ 値が小さいほど，先に起動します。また，値が大きいほど，先に停止します。なお，Enterprise Bean に対して同じ値を指定した場合，それらの間の起動順序は保証されません。

プロパティの設定項目については，次の個所を参照してください。

- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.4.1 Session Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.5.1 Entity Bean 属性ファイルの指定内容」
- マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.6.1 MessageDrivenBean 属性ファイルの指定内容」

9.15.3 サーブレットと JSP の起動順序の設定

サーブレット，JSP が含まれる WAR の起動順序を設定します。

(1) 編集する属性ファイル

WAR 属性ファイル

(2) 編集する属性ファイルの取得と属性の設定

- **属性ファイルの取得**

  次のコマンドを実行して，次の WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name account -type war -resname account_war -c C:¥home¥account_war.xml
  ```

- **属性の設定**

  次のコマンドを実行して，WAR 属性ファイルの値を反映します。

実行形式

```
cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type war -resname ＜WAR表示名＞ -c ＜WAR属性ファイルパス＞
```

実行例

```
cjsetappprop MyServer -name account -type war -resname account_war -c C:¥home¥account_war.xml
```

(3) 編集する属性設定項目

サーブレット，JSP が含まれる WAR の起動順序設定項目を次に示します。

項目	必須	対応するタグ名
開始・停止順※	△	<start-order>

（凡例） △：任意

注※ 値が小さいほど，先に起動します。また，値が大きいほど，先に停止します。なお，サーブレットと JSP に対して同じ値を指定した場合，それらの間の起動順序は保証されません。

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.7.1 WAR 属性ファイルの指定内容」を参照してください。

9.16　サーブレットと JSP のエラー通知の設定

9.16.1　設定方法

次のタイミングでエラーが発生した場合に，エラーを通知して J2EE アプリケーションの開始処理を中止するかどうかを指定します。

- J2EE アプリケーションの開始時にロードするように設定されている（<load-on-startup>が設定されている）サーブレットまたは JSP の初期化処理中にエラーが発生した場合
- タグライブラリ解析時にエラーが発生した場合

(1)　編集する属性ファイル

WAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次のコマンドを実行して，次の WAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjgetappprop MyServer -name account -type war -resname account_war -c C:¥home¥account_war.xml
  ```

- 属性の設定

  次のコマンドを実行して，WAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ -name <J2EEアプリケーション名> -type war -resname <WAR表示名> -c <WAR属性ファイルパス>
  ```

  実行例

  ```
  cjsetappprop MyServer -name account -type war -resname account_war -c C:¥home¥account_war.xml
  ```

(3)　編集する属性設定項目

サーブレット，JSP が含まれる WAR のエラー通知の設定項目を次に示します。

項目	必須	対応するタグ名
開始時のエラー通知有無	△	<start-notify-error>

（凡例） △：任意

(4) 注意事項

- Web アプリケーション初期化失敗時のエラー通知の設定可否

  J2EE サーバモードでは，Web アプリケーションの初期化で失敗した場合に，J2EE アプリケーションの開始処理を継続するか，エラーを通知して J2EE アプリケーションの開始処理を中止するかを設定できます。ただし，Web アプリケーションを利用不可とする必要があるエラーが発生した場合は，エラー通知の設定はできません。エラー通知の設定内容に関係なく，J2EE アプリケーションの開始処理が中止されます。

  ここでは，Web アプリケーションの初期化失敗の内容とエラー通知設定の可否について説明します。Web アプリケーションの初期化中に発生するエラーの内容と，J2EE アプリケーションの開始処理を継続するか，またはエラーとして中止するかのエラー通知の設定可否を次の表に示します。

表 9-5 Web アプリケーションの初期化失敗の内容とエラー通知の設定可否

項目	Web アプリケーション初期化失敗の内容	エラー通知の設定可否
JSP 用テンポラリディレクトリ	JSP 用テンポラリディレクトリのアクセス権がないため，ディレクトリの生成に失敗した場合。	不可
	JSP 用テンポラリディレクトリのアクセス権がないため，ディレクトリの削除に失敗した場合。	可
web.xml	web.xml の解析でエラーが発生した場合。	不可
フィルタ	web.xml の filter-class 要素に指定したフィルタクラスまたはそれが依存するクラスを見つけることができなかった場合。	不可
	web.xml の filter-class 要素に，次に示す不正なフィルタクラスが指定された場合。 1. javax.servlet.Filter インタフェースを実装していないクラス 2. 引数を取らないコンストラクタを持っていないクラス	不可
	フィルタクラスの初期化で例外が発生した場合。	不可
リスナ	web.xml の listener-class 要素に指定したリスナクラス，またはそれが依存するクラスを見つけることができなかった場合。	不可
	web.xml の listener-class 要素に，次に示す不正なリスナクラスが指定された場合。 1. 次に示すインタフェースを一つも実装していないクラス javax.servlet.ServletContextListener javax.servlet.ServletContextAttributeListener javax.servlet.ServletRequestListener javax.servlet.ServletRequestAttributeListener javax.servlet.http.HttpSessionListener javax.servlet.http.HttpSessionAttributeListener	不可

項目	Web アプリケーション初期化失敗の内容	エラー通知の設定可否
	2. 引数を取らないコンストラクタを持っていないクラス	
	リスナクラスの初期化で例外が発生した場合。	不可
web.xml で<load-on-startup>を指定したサーブレット	web.xml の servlet-class 要素に指定したサーブレットクラス，またはそれが依存するクラスを見つけることができなかった場合。	可
	サーブレットの初期化で例外が発生した場合。	可
web.xml で<load-on-startup>を指定した JSP	web.xml の jsp-file 要素に指定した JSP ファイルを見つけることができなかった場合。	可
	JSP 用テンポラリディレクトリにサブディレクトリを作成する権限がないため，JSP から Java ソースファイルを生成できなかった場合。	可
	JSP 用テンポラリディレクトリのサブディレクトリにアクセスする権限がないため，JSP から Java ソースファイルを生成できなかった場合。	可
	JSP から Java ソースファイルを生成する際にエラーが発生した場合。	可
	JSP から生成されたサーブレットのソースコードのコンパイルで，エラーが発生した場合。	可
	Servlet2.3 以前の Web アプリケーションで，web.xml の<taglib>タグで taglib をマッピングしていない，かつ JSP の taglib ディレクティブの uri 属性に絶対 URI を指定した場合。	可
	JSP ドキュメントの解析中にエラーが発生した場合。	可
	JSP の初期化で例外が発生した場合。	可
タグライブラリ（<load-on-startup>を指定した JSP の延長で実行される場合）	TLD ファイルの読み込みができなかった場合。	可
	TLD ファイルの解析中にエラーが発生した場合。	可
	タグライブラリバリデータクラスまたはそれが依存するクラスを見つけることができなかった場合。	可
	次に示す不正なタグライブラリバリデータクラスが指定された場合。 1. javax.servlet.jsp.tagext.TagLibraryValidator クラスを継承していないクラス 2. 引数を取らないコンストラクタを持っていないクラス	可
	タグライブラリバリデータクラスの初期化で例外が発生した場合。	可
	タグライブラリバリデータによる JSP の検証で例外が発生した場合。	可
	TagExtraInfo クラス，またはそれが依存するクラスを見つけることができなかった場合。	可
	不正な TagExtraInfo クラスが指定された場合。不正な TagExtraInfo クラスとは，Tei-class 要素または teiclass 要素に指定するクラスが，次のどれかの条件に一致するクラスを指す。 • TagExtraInfo クラスを継承していない • インタフェースや abstract クラスである • public なコンストラクタを持っていない	可

項目	Web アプリケーション初期化失敗の内容	エラー通知の設定可否
	TLD ファイルの tei-class 要素または teiclass 要素で定義したクラスの初期化で例外が発生した場合。	可
	タグライブラリバリデータクラスが JSP ページの検証で発見したエラーを報告した場合。	可
	TagExtraInfo クラスが属性の検証で発見したエラーを報告した場合。	可
	TagExtraInfo クラスによる属性の検証で例外が発生した場合。	可
	TLD ファイルの function-class 要素で定義したクラスを見つけることができなかった場合。	可
	TLD ファイルの function-class 要素で定義したクラスのロードで例外が発生した場合。	可
	TLD ファイルの function-signature 要素で定義した parameter クラスを見つけることができなかった場合。	可
	TLD ファイルの function-signature 要素で定義したメソッドを見つけることができなかった場合。	可
	TLD ファイルの function-signature 要素で定義した parameter クラスのロード，または function-class 要素で指定したクラスへのアクセス時に例外が発生した場合。	可
	TLD ファイルの type 要素で定義したクラスを見つけることができなかった場合。	可
	TLD ファイルの type 要素で定義したクラスのロードで例外が発生した場合。	可

9.17　インターセプタの設定

9.17.1　設定方法

インターセプタの設定方法について説明します。アプリケーションサーバでインターセプタを使用する場合は，EJB-JAR の属性として設定してください。

(1)　編集する属性ファイル

EJB-JAR 属性ファイル

(2)　編集する属性ファイルの取得と属性の設定

- 属性ファイルの取得

  次に示すコマンドを実行して EJB-JAR 属性ファイルを取得します。

  実行形式

  ```
  cjgetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type ejb -resname ＜EJB-JAR表示名＞ -c ＜EJB-JAR属性ファイルパス＞
  ```

  実行例

  ```
  cjgetappprop MyServer -name adder -type ejb -resname adder -c C:¥home¥adder_ejb.xml
  ```

- 属性の設定

  次に示すコマンドを実行して，EJB-JAR 属性ファイルの値を反映します。

  実行形式

  ```
  cjsetappprop ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -type ejb -resname ＜EJB-JAR表示名＞ -c ＜EJB-JAR属性ファイルパス＞
  ```

  実行例

  ```
  cjsetappprop MyServer -name adder -type ejb -resname adder -c C:¥home¥adder_ejb.xml
  ```

注意事項

開始状態のアプリケーションに含まれる EJB-JAR 属性ファイルの取得はできますが，定義された関連情報の反映はできません。

(3)　編集する属性設定項目

インターセプタの設定項目（<interceptor-binding>）を次に示します。

項目	必須	対応するタグ名
説明	△	<description>

項目	必須	対応するタグ名
インターセプタの EJB 名	○	<ejb-name>
インターセプタクラスのクラス名	△	<interceptor-class>
インターセプタの実行順序の設定（インターセプタクラスのクラス名）	△	<interceptor-order>-<interceptor-class>
デフォルトインターセプタの呼び出し抑止の設定	△	<exclude-default-interceptors>
クラスレベルインターセプタの呼び出し抑止の設定	△	<exclude-class-interceptors>
EJB のビジネスメソッド名	△	<named-method>-<method-name>
メソッドの引数	△	<named-method>-<method-params>-<method-param>

（凡例） ○：必須　△：任意

プロパティの設定項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」の「3.3.1　EJB-JAR 属性ファイルの指定内容」を参照してください。

インターセプタの設定方法や注意事項については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」の「2.15　インターセプタの使用」を参照してください。

9.18　そのほかのプロパティの設定

J2EE アプリケーションの作成およびカスタマイズで設定する，そのほかのプロパティ項目については，マニュアル「アプリケーションサーバ リファレンス 定義編(アプリケーション／リソース定義)」を参照してください。

10 J2EE アプリケーションの実行

この章では，サーバ管理コマンドを使用した J2EE アプリケーションの実行について説明します。

10.1　J2EE アプリケーションの実行の概要

J2EE アプリケーションとして必要なプロパティや動作の定義が終わったら，J2EE アプリケーションは実行できます。

J2EE アプリケーションの実行の概要について次に示します。

表 10–1　J2EE アプリケーションの実行の概要

設定項目	内容	J2EE アプリケーションの形式		参照先
		アーカイブ形式	展開ディレクトリ形式	
J2EE アプリケーションの開始と停止	J2EE アプリケーションを，通常開始または JSP をコンパイルして開始します。 J2EE アプリケーションを，通常停止または強制停止します。	○	○	10.2
J2EE アプリケーションの一覧の参照	インポートされているアプリケーションの一覧を参照します。	○	○	10.3
J2EE アプリケーションの削除	J2EE アプリケーションを削除します。	○	△※1	10.4
J2EE アプリケーションの入れ替え	J2EE サーバ上の，次の J2EE アプリケーションを入れ替えます。 • アーカイブ形式のアプリケーション • 展開ディレクトリ形式のアプリケーション	△※2	△※2	10.5
J2EE アプリケーション名の変更	J2EE アプリケーションの名前を変更します。	○	○	10.6
RMI-IIOP スタブとインタフェースの取得	J2EE アプリケーションの RMI-IIOP スタブおよびインタフェースを取得します。	○	○	10.7
トランザクション一覧の参照	トランザクションの情報の一覧を参照します。 J2EE サーバが稼働中の場合は，稼働状況や，開始してからの経過時間などのトランザクションの情報を表示します。 J2EE サーバが停止中の場合は，ステータスファイルにある，仕掛かり中のまま残っている未決着トランザクションの状況を表示します。	○	○	10.8

（凡例）　○：実行できる　△：実行条件がある

注※1　アプリケーションからコンポーネントの削除はできません。

なお，フィルタの追加は，アプリケーションの属性変更であるため実行できます。

注※2　J2EE アプリケーションの入れ替えは，次のコマンドを使用します。

- アーカイブ形式の場合　cjreplaceapp コマンド
- 展開ディレクトリ形式の場合　cjreloadapp コマンド

10.2　J2EE アプリケーションの開始と停止

ここでは，J2EE アプリケーションの開始と停止について説明します。

10.2.1　J2EE アプリケーションの開始

ここでは，次の J2EE アプリケーションの開始方法について説明します。

- 通常開始
- JSP をコンパイルして開始

(1)　J2EE アプリケーションの通常開始

次に示すコマンドを実行して J2EE アプリケーションを開始します。J2EE アプリケーションにリソースアダプタが含まれている場合，J2EE アプリケーションに含まれるすべてのリソースアダプタも自動で開始されます。

実行形式

```
cjstartapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞
```

実行例

```
cjstartapp MyServer -name account
```

cjstartapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始)」を参照してください。

注意事項

- J2EE アプリケーションが，J2EE リソースアダプタを参照している場合，J2EE リソースアダプタを開始してから，J2EE アプリケーションを開始してください。
- J2EE アプリケーションに含まれるリソースアダプタは，開始順序を制御はできません。
- ほかの J2EE アプリケーションに含まれる Enterprise Bean を呼び出す場合，呼び出し先の J2EE アプリケーションを開始してから，呼び出し元の J2EE アプリケーションを開始してください。

(2)　JSP をコンパイルして開始

J2EE アプリケーションに含まれるすべての JSP ファイルのコンパイルを実行して，J2EE アプリケーションを開始します。

次に示すコマンドを実行します。

実行形式

```
cjstartapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -jspc
```

実行例

```
cjstartapp MyServer -name App1 -jspc
```

cjstartapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

注意事項

アーカイブ形式のアプリケーションの場合，アプリケーションの開始時に JSP 事前コンパイルを実施して生成された JSP コンパイル結果は，アプリケーションの停止時に削除されます。

アプリケーション開始時の JSP 事前コンパイルで生成した JSP コンパイル結果をアプリケーションの停止後も利用する場合，次の手順を実行します。

1. JSP の事前コンパイルを指定してアプリケーションを開始します。
2. アプリケーションをエクスポートします。
3. リデプロイ機能などを使用して，JSP コンパイル結果を含むアプリケーションに入れ替えます。
   J2EE アプリケーションの入れ替えについては，「10.5　J2EE アプリケーションの入れ替え」を参照してください。

また，アプリケーション開始時の JSP 事前コンパイルを実行したあとに，アプリケーションの開始に失敗した場合，アプリケーションは停止し，JSP コンパイル結果は削除されます。アプリケーション開始時の JSP 事前コンパイルを実行する前に，アプリケーションが正常に開始できることを確認してください。

10.2.2　J2EE アプリケーションの停止

ここでは，J2EE アプリケーションの通常停止と強制停止について説明します。

(1)　J2EE アプリケーションの通常停止

次に示すコマンドを実行して，J2EE アプリケーションを通常停止します。J2EE アプリケーションにリソースアダプタが含まれている場合，J2EE アプリケーションに含まれるすべてのリソースアダプタも自動で停止されます。

実行形式

```
cjstopapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞
```

実行例

```
cjstopapp MyServer -name account
```

cjstopapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstopapp（J2EE アプリケーションの停止）」を参照してください。

注意事項

- J2EE アプリケーションが J2EE リソースアダプタとしてデプロイしたリソースアダプタを参照している場合，J2EE アプリケーションを停止してから，J2EE リソースアダプタを停止してください。
- J2EE アプリケーションに含まれるリソースアダプタは，停止順序を制御はできません。
- ほかの J2EE アプリケーションに含まれる Enterprise Bean を呼び出す場合，呼び出し元の J2EE アプリケーションを停止してから，呼び出し先の J2EE アプリケーションを停止してください。
- アーカイブ形式のアプリケーションの場合，アプリケーション開始時の JSP 事前コンパイルを実行して生成されたコンパイル結果は，アプリケーションの停止時に削除されます。

(2) J2EE アプリケーションの強制停止

J2EE アプリケーションを強制停止します。

J2EE アプリケーションの強制停止には，次の二つがあります。

- J2EE アプリケーションの通常停止を実行して，J2EE アプリケーションが停止しなかった場合に強制停止を実行します。
- 停止しなかった場合に自動に強制停止を実行するオプションを指定して，J2EE アプリケーションの通常停止を実行します。

それぞれの強制停止の手順について，次に示します。

(a) J2EE アプリケーションの通常停止を実行して J2EE アプリケーションが停止しなかった場合に強制停止を実行する

「(1) J2EE アプリケーションの通常停止」の手順で，J2EE アプリケーションの通常停止を実行しても J2EE アプリケーションが正常に停止しなかった場合，次の手順で J2EE アプリケーションを強制停止できます。

なお，J2EE アプリケーションの強制停止は，通常停止を実行しても J2EE アプリケーションが正常に停止しなかった場合にだけ実行してください。

次に示すコマンドを実行して J2EE アプリケーションを強制停止します。

実行形式

```
cjstopapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -cancel
```

実行例

```
cjstopapp MyServer -name account -cancel
```

cjstopapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstopapp（J2EE アプリケーションの停止)」を参照してください。

(b) J2EE アプリケーションの自動強制停止オプションを指定して，通常停止を実行する

J2EE アプリケーションの通常停止を実行しても J2EE アプリケーションが正常に停止しなかった場合，タイムアウト時間が経過すると自動的に強制停止を実行できます。

次のコマンドを実行して，通常停止しない J2EE アプリケーションを自動的に強制停止します。

実行形式

```
cjstopapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞ -t ＜タイムアウト時間（秒）＞ -force
```

実行例

```
cjstopapp MyServer -name account -t 120 -force
```

cjstopapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstopapp（J2EE アプリケーションの停止)」を参照してください。

10.3　J2EE アプリケーションの一覧の参照

次のコマンドを実行して，インポートされている J2EE アプリケーションの一覧を参照します。

J2EE アプリケーションが展開ディレクトリ形式の場合，アプリケーションディレクトリのパスを表示します。

実行形式

```
cjlistapp ［＜サーバ名称＞］
```

実行例

```
cjlistapp MyServer
```

cjlistapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlistapp（アプリケーションの一覧表示)」を参照してください。

10.4　J2EE アプリケーションの削除

J2EE アプリケーションを削除します。

次に示すコマンドを実行して J2EE アプリケーションを削除します。

実行形式

```
cjdeleteapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜J2EEアプリケーション名＞
```

実行例

```
cjdeleteapp MyServer -name account
```

cjdeleteapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjdeleteapp（J2EE アプリケーションの削除)」を参照してください。

注意事項

展開ディレクトリ形式の J2EE アプリケーションを削除する場合，削除対象の J2EE アプリケーションのアプリケーションディレクトリは削除されません。J2EE サーバへの登録が解除されます。

10.5　J2EE アプリケーションの入れ替え

J2EE サーバモード上で動作する J2EE アプリケーションの入れ替えには，次の方法があります。

- J2EE アプリケーションを再デプロイします。
  J2EE アプリケーションを J2EE サーバから削除し，別の J2EE アプリケーションを入れ替え，開始します。
- リデプロイ機能を使用します。
  J2EE アプリケーションがアーカイブ形式の場合に使用できます。
- リロード機能を使用します。
  展開ディレクトリ形式の場合に使用できます。

この節では，次の J2EE アプリケーションの入れ替えについて説明します。

- リデプロイ機能を使用した，アーカイブ形式の J2EE アプリケーションの入れ替え
- リロード機能を使用した，展開ディレクトリ形式の J2EE アプリケーションの入れ替え

10.5.1　アーカイブ形式のアプリケーション

リデプロイ機能によって，J2EE サーバにインポートしたアーカイブ形式の J2EE アプリケーションを，別の J2EE アプリケーションに入れ替える手順を次に示します。

1. 更新する Java プログラムをコンパイルします。
2. EJB-JAR，WAR ファイルを再生成します。
3. EAR を再生成します。
4. リデプロイコマンドを実行します。

次のリデプロイコマンドを実行して，アーカイブ形式の J2EE アプリケーション（EAR ファイル）を入れ替えます。

実行形式

```
cjreplaceapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜アプリケーション名＞ -f ＜EARファイルのパス＞ ［-t ＜タイムアウト時間（秒）＞］ ［-replaceDD］
```

実行例

```
cjreplaceapp MyServer -name App1 -f App1.ear -t 120
```

J2EE アプリケーションの入れ替え時には，J2EE アプリケーションのすべての属性を引き継ぐことも，ランタイム属性（属性ファイルの独自の定義）だけを引き継ぐこともできます。ランタイム属性だけを引き継ぎたい場合は-replaceDD オプションを指定します。

cjreplaceapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjreplaceapp（アプリケーションの入れ替え）」を参照してください。

注意事項

- この操作は J2EE アプリケーションが開始，停止のどちらの状態であっても実行できます。開始状態の J2EE アプリケーションを入れ替える場合，cjreplaceapp コマンドを実行すると，処理の中で J2EE アプリケーションはいったん停止され，入れ替え後に再開始されます。

  なお，開始中状態の J2EE アプリケーションが正常に停止されなかった場合，J2EE アプリケーションは強制停止されます。-t オプションにタイムアウト時間を指定すると，J2EE アプリケーションの強制停止に移行するまでの時間を設定できます。-t オプションにタイムアウト時間を指定しなかった場合，強制停止に移行するまでの時間は 60 秒です。
- JSP 事前コンパイルを実行したアプリケーションの入れ替えで，入れ替え後も JSP 事前コンパイル機能を使用する場合，入れ替えるアプリケーションに JSP コンパイル結果が含まれていなければなりません。入れ替えるアプリケーションに JSP コンパイル結果が含まれていない場合，Web アプリケーション単位の JSP 事前コンパイルを実行し，入れ替えるアプリケーションに JSP コンパイル結果を含めてください。
- cosminexus.xml を含むアプリケーションにリデプロイ機能を使用した場合，入れ替え前の cosminexus.xml の定義情報を破棄してデフォルト値に戻したあと，入れ替え後の cosminexus.xml の定義情報を上書きします。そのため，実行環境でサーバ管理コマンドを使って変更したアプリケーションサーバ独自の定義情報は失われます。

10.5.2 展開ディレクトリ形式のアプリケーション

リロード機能を使用する場合は，アプリケーションの再生成（EAR ファイル作成）が不要となり，直接クラスファイルを置き換えるだけでよいため，アプリケーションの入れ替えが容易になります。

リロード機能は，展開ディレクトリ形式を使用し，開始状態およびリロードに失敗して停止状態にある J2EE アプリケーションに対してだけ実行できます。

リロード機能を実現するには，次の二つの手段があります。

- J2EE サーバが更新を自動的に検知することによるリロード機能
- コマンドの実行によるリロード機能

ここでは，コマンドの実行によるリロード機能について説明します。

リロード機能によって，J2EE アプリケーションに入れ替える手順を次に示します。

1. 更新する Java プログラムをコンパイルします。

2. リロードコマンドを実行します。

次のリロードコマンドを実行して，展開ディレクトリ形式の J2EE アプリケーションを入れ替えます。

(1) 実行形式

```
cjreloadapp ［サーバ名］ -name アプリケーション名 ［-t 強制リロード開始までのタイムアウト時間］
```

(2) 実行例

```
cjreloadapp MyServer -name App1
```

cjreloadapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjreloadapp（アプリケーションのリロード)」を参照してください。

リロード機能の詳細および更新検知によるリロード機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「18.8　J2EE アプリケーションの更新検知とリロード」を参照してください。

(3) 注意事項

- デフォルトの設定（app）では，リロード機能は有効になっています。なお，更新検知によるリロード機能を使用するためには，プロパティ設定ファイル（usrconf.properties）のキーを，次のように設定してください。
  - リロード機能の適用範囲の設定

    ejbserver.deploy.context.reload_scope=app
  - 更新検知インターバルの設定

    ejbserver.deploy.context.check_interval=1

  J2EE アプリケーションの更新検知とリロードの設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「18.8.12　J2EE アプリケーションの更新検知とリロードの設定」を参照してください。
- リソースアダプタを含む J2EE アプリケーションの場合，RAR ファイルはアーカイブ形式で格納されています。アプリケーションディレクトリ下にある RAR ファイルが変更されてもリロードは実行されません。
- cosminexus.xml を含むアプリケーションにリロード機能を使った場合，cosminexus.xml の定義情報についてはリロードされません。

10.6　J2EE アプリケーション名の変更

J2EE サーバにインポートした J2EE アプリケーションの名称を変更します。

J2EE アプリケーションの名称を変更するには，J2EE アプリケーションを停止しておく必要があります。J2EE アプリケーションを停止する方法については「10.2.2　J2EE アプリケーションの停止」を参照してください。

次に示すコマンドを実行して J2EE アプリケーション名を変更します。

実行形式

```
cjrenameapp ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜アプリケーション名＞ -newname ＜変更後のアプリケーション名＞
```

実行例

```
cjrenameapp MyServer -name App1 -newname App2
```

cjrenameapp コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjrenameapp（アプリケーション名の変更)」を参照してください。

10.7 RMI-IIOP スタブとインタフェースの取得

10.7.1 実行形式と説明

J2EE サーバにインポートした J2EE アプリケーションの RMI-IIOP スタブとインタフェースを取得します。

J2EE サーバにインポートした J2EE アプリケーションの RMI-IIOP スタブおよびインタフェースは，J2EE アプリケーションを一度も開始していない状態では取得できません。

また，J2EE アプリケーション内に，リモートインタフェースを持つ Enterprise Bean が存在しない場合，RMI-IIOP スタブおよびインタフェースの取得はエラーになります。

次に示すコマンドを実行して RMI-IIOP スタブおよびインタフェースを取得します。

(1) 実行形式

```
cjgetstubsjar ［＜サーバ名称＞］ ［-nameserver ＜プロバイダURL＞］ -name ＜アプリケーション名＞ -d ＜RMI-IIOPスタブおよびインタフェースの格納パス＞
```

(2) 実行例

```
cjgetstubsjar MyServer -name App1 -d temp
```

(3) 注意事項

- 次の条件を満たすときだけ，この機能を使用できます。
  - J2EE サーバに旧バージョンで作成した J2EE アプリケーションが存在する状態で，アプリケーションサーバをアップグレードインストールした場合
  - cjrenameapp コマンドで名称を変更していない場合
- 次の場合は，index.html による RMI-IIOP スタブおよびインタフェースの取得はできません。
  - 実行時情報を含む J2EE アプリケーションまたは実行時情報を含まない J2EE アプリケーションのどちらをインポートした場合も，実行できません。
  - 新規に J2EE アプリケーションを作成した場合
  - cjrenameapp コマンドで名称を変更した場合
- application.xml を省略したアプリケーションの名前を変更した場合，application.xml が作成されます。そのため，アプリケーション名を変更した J2EE アプリケーションは application.xml のあるアプリケーションとなります。

cjgetstubsjar コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjgetstubsjar（アプリケーションの RMI-IIOP スタブおよびインタフェースの取得)」を参照してください。

10.8　トランザクション一覧の参照

10.8.1　実行形式と説明

トランザクション情報の一覧を参照します。稼働中の J2EE サーバで動作しているトランザクション情報，および停止中の J2EE サーバに残っている未決着トランザクション情報を参照できます。

(1)　稼働中の J2EE サーバでのトランザクション情報の表示

稼働状況や開始してからの経過時間など，トランザクションの情報の一覧を参照します。

(a)　実行形式

```
cjlisttrn ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ ［-gid <グローバルトランザクションID>］ ［-pending］ ［-time <経過時間>］ ［-bqual］
```

(b)　実行例

トランザクションに関連しているすべての Xid のブランチ ID を表示する例

```
cjlisttrn MyServer -bqual
```

cjlisttrn コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlisttrn（稼働中の J2EE サーバのトランザクション情報の表示)」を参照してください。

(2)　停止中の J2EE サーバでのトランザクション情報の表示

停止中の J2EE サーバのステータスファイルに残っている，未決着トランザクション情報の一覧を参照します。

次に示すコマンドを実行してトランザクションの一覧を参照します。

(a)　実行形式

```
cjlisttrnfile ［<サーバ名称>］ ［-nameserver <プロバイダURL>］ ［-gid <グローバルトランザクションID>］ ［-bqual］
```

(b)　実行例

指定のグローバルトランザクション ID に関する情報を表示する例

```
cjlisttrnfile MyServer -gid d13800010000000000000000000000fefb57e6480000000000000001
```

cjlisttrnfile コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjlisttrnfile（停止中の J2EE サーバのトランザクション情報の表示)」を参照してください。

付録

付録 A　Server Plug-in からの移行

旧バージョンで使用していた Server Plug-in での操作は，運用管理ポータル，標準 DD，cosminexus.xml などを代替手段として利用できます。

ここでは，次の Server Plug-in の操作について，利用できる代替手段を説明します。

- 論理サーバの運用管理
  論理サーバの起動や停止，稼働状況の表示などの操作について説明します。
- リソースアダプタの操作
  リソースアダプタのインポートやエクスポート，デプロイなどの操作について説明します。
- J2EE アプリケーションの操作
  J2EE アプリケーションのインポートやエクスポート，開始や停止などの操作について説明します。
- J2EE アプリケーションに含まれるリソースアダプタの設定
  リソースアダプタを含む J2EE アプリケーションのインポート，開始や停止などの操作について説明します。
- J2EE アプリケーションのプロパティ設定
  次に示す項目について説明します。
  - Enterprise Bean のプロパティ設定項目
  - サーブレットと JSP のプロパティ設定項目
  - J2EE アプリケーション（共通）のプロパティ設定項目

なお，表中の参照先を示す「参照先マニュアル」に記載したマニュアル名の「アプリケーションサーバ」は省略しています。それぞれについて説明します。

付録 A.1　論理サーバの運用管理

Server Plug-in に代わって，論理サーバの運用管理は次に示す手段が利用できます。

表 A-1　論理サーバの運用管理の操作方法および参照先

Server Plug-in の操作		代替手段	参照先マニュアル	参照個所
操作概要	操作詳細			
論理サーバの起動	開始オプションの設定	運用管理ポータル	運用管理ポータル操作ガイド	10.8.2
	ドメイン単位の論理サーバの一括起動	運用管理ポータル	運用管理ポータル操作ガイド	11.3.2
	ホスト単位の論理サーバの一括起動	運用管理ポータル	運用管理ポータル操作ガイド	11.2.2

Server Plug-in の操作		代替手段	参照先マニュアル	参照個所
操作概要	操作詳細			
	個別の論理サーバの起動	運用管理ポータル	運用管理ポータル操作ガイド	11.4.2，11.5.2，11.6.2，11.7.2，11.8.2，11.9.2，11.10.2，11.10.6，11.11.2，11.12.2，11.12.6，11.13.2
	アプリケーション起動なしで J2EE サーバを起動	運用管理ポータル	運用管理ポータル操作ガイド	11.9.2
論理サーバの停止	ドメイン単位の論理サーバの一括停止	運用管理ポータル	運用管理ポータル操作ガイド	11.3.3
	ホスト単位の論理サーバの一括停止	運用管理ポータル	運用管理ポータル操作ガイド	11.2.3
	個別の論理サーバの停止	運用管理ポータル	運用管理ポータル操作ガイド	11.4.3，11.5.3，11.6.3，11.7.3，11.8.3，11.9.3，11.10.3，11.10.7，11.11.3，11.12.3，11.12.7，11.13.3
	論理 J2EE サーバの強制停止	運用管理ポータル	運用管理ポータル操作ガイド	11.10.7
論理サーバの稼働状態表示	運用管理ドメイン内のすべての論理サーバの稼働状態	運用管理ポータル	運用管理ポータル操作ガイド	11.3.1

付録 A.2　リソースアダプタの操作

リソースアダプタの操作は，Server Plug-in に代わって次の手段が利用できます。

表 A-2　リソースアダプタの操作方法および参照先

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
リソースアダプタのインポート	運用管理ポータル	運用管理ポータル操作ガイド	12.4.4
リソースアダプタのプロパティ定義	運用管理ポータル	運用管理ポータル操作ガイド	12.4.5
リソースアダプタのデプロイ	なし（インポート後に自動的に実行されます）。	－	－

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
リソースアダプタの接続テスト	運用管理ポータル	運用管理ポータル操作ガイド	12.4.3
リソースアダプタの開始	運用管理ポータル	運用管理ポータル操作ガイド	12.4.1
リソースアダプタの停止	運用管理ポータル	運用管理ポータル操作ガイド	12.4.2
リソースアダプタの削除	運用管理ポータル	運用管理ポータル操作ガイド	12.4.6
リソースアダプタの稼働状態一覧	運用管理ポータル	運用管理ポータル操作ガイド	12.4.1
リソースアダプタのエクスポート	コマンド（cjexportrar）	リファレンス コマンド編	cjexportrar（リソースアダプタのエクスポート）
Connector 属性のインポートとエクスポート※	運用管理ポータル	運用管理ポータル操作ガイド	12.4.5

（凡例）

－：該当なし。

注※

運用管理ポータルの［リソースアダプタの Connector 属性ファイル編集］画面で，Connector 属性を指定，参照できます。

付録 A.3　J2EE アプリケーションの操作

J2EE アプリケーションの操作は，Server Plug-in に代わって次の手段が利用できます。

表 A-3　J2EE アプリケーションの操作方法および参照先

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
J2EE アプリケーションのインポート（アーカイブ形式）	運用管理ポータル	運用管理ポータル操作ガイド	12.3.3
J2EE アプリケーションのインポート（展開ディレクトリ形式）	運用管理ポータル	運用管理ポータル操作ガイド	12.3.4
J2EE アプリケーションの開始	運用管理ポータル	運用管理ポータル操作ガイド	12.3.1
J2EE アプリケーションの通常停止	運用管理ポータル	運用管理ポータル操作ガイド	12.3.2
J2EE アプリケーションの強制停止	コマンド（cjstopapp）	リファレンス コマンド編	cjstopapp（J2EE アプリケーションの停止）
J2EE アプリケーションの一覧の参照	運用管理ポータル	運用管理ポータル操作ガイド	12.3.1

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
J2EE アプリケーションの削除	運用管理ポータル	運用管理ポータル操作ガイド	12.3.5
J2EE アプリケーションの入れ替え（アーカイブ形式）	運用管理ポータル	運用管理ポータル操作ガイド	4.3
J2EE アプリケーションの入れ替え（展開ディレクトリ形式）	運用管理ポータル	運用管理ポータル操作ガイド	4.3
J2EE アプリケーション名の変更	コマンド（cjrenameapp）	リファレンス コマンド編	cjrenameapp（アプリケーション名の変更）
RMI-IIOP スタブとインタフェースの取得	コマンド（cjgetstubsjar）	リファレンス コマンド編	cjgetstubsjar（アプリケーションの RMI-IIOP スタブおよびインタフェースの取得）
アプリケーション統合属性のインポート	コマンド（cjsetappprop）※	リファレンス コマンド編	cjsetappprop（アプリケーションの属性設定）
アプリケーション統合属性のエクスポート	コマンド（cjgetappprop）※	リファレンス コマンド編	cjgetappprop（アプリケーションの属性の取得）
J2EE アプリケーションのエクスポート	コマンド（cjexportapp）	リファレンス コマンド編	cjexportapp（J2EE アプリケーションのエクスポート）

注※

cosminexus.xml を利用してアプリケーションの情報を管理することを推奨します。cosminexus.xml を含んだ J2EE アプリケーションの詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」の「16. アプリケーションの属性管理」を参照してください。

付録 A.4 J2EE アプリケーションに含まれるリソースアダプタの設定

J2EE アプリケーションに含まれるリソースアダプタの設定は，Server Plug-in に代わって次の手段が利用できます。

表 A-4 リソースアダプタの設定の操作方法および参照先

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
リソースアダプタを含む J2EE アプリケーションのインポート	運用管理ポータル	運用管理ポータル操作ガイド	12.3.3
リソースアダプタのプロパティ定義	cosminexus.xml (rar タグなど)	リファレンス 定義編 (アプリケーション／リソース定義)	2.2
リソースアダプタの接続テスト	コマンド（cjtestres）	リファレンス コマンド編	cjtestres（リソースの接続テスト）

Server Plug-in の操作	代替手段	参照先マニュアル	参照個所
リソースアダプタを含む J2EE アプリケーションの開始	運用管理ポータル	運用管理ポータル操作ガイド	12.3.1
リソースアダプタを含む J2EE アプリケーションの停止	運用管理ポータル	運用管理ポータル操作ガイド	12.3.2
Connector 属性のインポートとエクスポート	コマンド（cjgetappprop）	リファレンス コマンド編	cjgetappprop（アプリケーションの属性の取得）
	コマンド（cjsetappprop）	リファレンス コマンド編	cjsetappprop（アプリケーションの属性設定）

付録 A.5　J2EE アプリケーションのプロパティ設定

ここでは，次に示す項目について説明します。

- Enterprise Bean のプロパティ設定項目
- サーブレットと JSP のプロパティ設定項目
- J2EE アプリケーション（共通）のプロパティ設定項目

(1)　Enterprise Bean のプロパティ設定項目

Enterprise Bean のプロパティ設定項目は，Server Plug-in に代わって次の手段が利用できます。

表 A-5　Enterprise Bean のプロパティ設定項目の操作方法および参照先

Server Plug-in の操作	代替手段	参照個所
ほかの Enterprise Bean のリファレンス定義（ejb-ref）	DD（ejb-jar.xml）	9.3.1
リソースアダプタのリファレンス定義（resource-ref）	cosminexus.xml，DD（ejb-jar.xml）	9.3.3
リソース環境のリファレンス定義（resource-env-ref）	cosminexus.xml，DD（ejb-jar.xml）	9.3.4
Message-driven Bean のメッセージ参照定義（message-ref）	cosminexus.xml	9.4
トランザクション属性の定義（trans-attribute）	DD（ejb-jar.xml）	9.5
Stateful Session Bean の実行時プロパティの設定	cosminexus.xml	9.10.1
Stateless Session Bean の実行時プロパティの設定	cosminexus.xml	9.10.2
Entity Bean の実行時プロパティの設定	cosminexus.xml	9.10.3
Message-driven Bean の実行時プロパティの設定	cosminexus.xml	9.10.4

(2)　サーブレットと JSP のプロパティ設定項目

サーブレットと JSP のプロパティ設定項目は，Server Plug-in に代わって次の手段が利用できます。

表 A-6　サーブレットと JSP のプロパティ設定項目の操作方法および参照先

Server Plug-in の操作	代替手段	参照個所
Enterprise Bean のリファレンス定義（ejb-ref）	DD（web.xml）	9.7.1
リソースアダプタのリファレンス定義（resource-ref）	cosminexus.xml，DD（web.xml）	9.7.3
リソース環境のリファレンス定義（resource-env-ref）	cosminexus.xml，DD（web.xml）	9.7.4
サーブレットと JSP のマッピング定義（url-pattern）	DD（web.xml）	9.8
フィルタのマッピング定義（filter-mapping）	DD（web.xml）	9.9.2
J2EE アプリケーションのコンテキストルート定義（context-root）	DD（application.xml）	9.11.1
Web アプリケーション単位での同時実行スレッド数制御の定義（thread-control）	cosminexus.xml	9.11.2
URL グループ単位での同時実行スレッド数制御の定義（urlgroup-thread-control）	cosminexus.xml	9.11.3
URL グループ単位の実行待ちリクエスト数の監視の定義（urlgroup-thread-control タグ - stats-monitor タグ - waiting-request-count タグ）	cosminexus.xml	9.11.4
デフォルトの文字エンコーディングの設定（http-request タグ - encoding タグ，http-response タグ - encoding タグ，jsp タグ - page-encoding タグ）	cosminexus.xml	9.12
サーブレットと JSP のエラー通知の設定	cosminexus.xml	9.16

(3)　J2EE アプリケーション（共通）のプロパティ設定項目

J2EE アプリケーション（共通）のプロパティ設定項目は，Server Plug-in に代わって次の手段が利用できます。

表 A-7　J2EE アプリケーション（共通）のプロパティ設定項目の操作方法および参照先

Server Plug-in の操作		代替手段	参照個所
操作概要	操作詳細		
CTM のスケジューリング	J2EE アプリケーション単位のスケジューリング	cosminexus.xml	9.14.1
	Stateless Session Bean 単位のスケジューリング	cosminexus.xml	9.14.2

Server Plug-in の操作		代替手段	参照個所
操作概要	操作詳細		
起動順序の設定	J2EE アプリケーションの起動順序の設定	cosminexus.xml	9.15.1
	Enterprise Bean の起動順序の設定	cosminexus.xml	9.15.2
	サーブレットと JSP の起動順序の設定	cosminexus.xml	9.15.3

索引

M

R

S

U

W

あ

へ

ほ

め

ゆ

ら

り

る

ろ